

AMENDMENTS TO CHAPTER 12 OF SILLIAC HANDBOOK

In September 1959 a magnetic tape transport, together with the control circuitry necessary for its operation by SILLIAC, was attached to the computer. This unit forms the nucleus of a magnetic tape backing store which, when complete, will consist of five tape transports, any four of which can be connected to the SILLIAC at one time.

Chapter 12 describes the tape store as it will be when it is complete. The following notes, with page references, describe those ways in which the present system differs from the five transport system described in Chapter 12.

Page

- 12-2 With only a single tape transport available, the computer cannot use the tape store while the reel mounted on the transport is being changed.
- 12-3 Only one of the four tape operations (record, playback, search, rewind) can take place at one time.
- 12-8 No other magnetic tape operations can take place while a search is in progress.
- 12-15 See above note for page 12-3.
- 12-17 While there is only a single transport the SILLIAC will not decode the address digits which specify the transport. Hence

Page

12-17
contd.

the transport may be referred to as number 0, 1, 2, or 3. However, programmers are advised to refer to it as number 0. The same number must be used throughout the program if it is to continue to operate after the full set of transports has been provided.

12-37

A program which uses a succession of tapes will have to wait while the tapes are changed on the single transport which is available.

12-38

All manual controls are available, except the jacks and plugs mentioned under (b). The controls are all mounted on the tape transport.

12-49

The technique of transferring information from one tape to another cannot be used while only a single transport is available. Hence if information is lost from a tape through a computer malfunction it can be recovered only by going back to the original source, i.e. to the paper tape from which the data originally came.

CHAPTER 12

THE MAGNETIC TAPE STORE

12.1 GENERAL DESCRIPTION

(This section is devoted to an introductory description of the physical nature of the tape store attached to SILLIAC and a definition of the terms used. Many of the statements made here are amplified in later sections.)

The SILLIAC magnetic tape store consists of a set of five tape transports and associated electronic circuitry. The magnetic tape itself is a strip of plastic material half an inch in width and about .0015 inch thick, on one side of which is a coating, about .0005 inch thick, of a magnetisable iron oxide. Information can be recorded on this tape as a series of binary digits, each of which is represented by the state of magnetisation of a small area of the oxide coating. A strip of tape, of length up to a maximum of 2,400 feet, is wound on a tape reel.

A transport is a mechanical device for transferring the magnetic tape from one reel to another at a controlled speed so that it passes over a read/write head which serves the alternative functions of recording information upon the tape or playing it back. The recording operation automatically erases any information previously recorded on the tape. The playback operation is non-destructive, i.e. an item of information, once

recorded, can be played back any number of times.

When a tape is mounted on a transport in the rewound condition, the take-up reel is empty, the tape, apart from the few feet extending through the loops and over the capstans between the reels, lying entirely on the main reel. When in this condition the main reel may be removed from the transport and replaced by another. This manual operation, which involves threading the new tape through the loops and capstans, takes about two minutes. Thus any one of an arbitrarily large number of tape reels may be mounted on each transport. Five transports are provided, even though only four can be connected to the SILLIAC at any one time, so that the tape reel may be changed on the idle unit while the other four are being actively used by the computer.

Starting from the rewound state, the transport may drive the tape from the main reel to the take-up reel. This direction of motion is called forward. The tape may also be driven in the reverse direction, which is called backward. Recording and playing back can take place while the tape is moving in either the forward or backward direction.

The tape may also be moved, in either direction, without a record or playback operation being performed, the movement being purely to locate the tape in a certain position. Most commonly this is performed by a search operation in which the transport and its ancillary electronic circuitry move the tape, in a selected direction, until a pre-selected piece of information is detected under the read/write head.

Such an operation does, of course, involve the reading of information from the tape, but as the information so read is not transferred to the SILLIAC we do not classify it as a playback operation. A search operation is initiated by an instruction carried out by the SILLIAC in the course of the program it is obeying, but, once initiated, it is carried out autonomously by the tape control circuitry, thus allowing the SILLIAC to proceed with other instructions while the tape is being positioned.

The fourth operation of which the transport is capable is fast rewind. In this operation the read/write head is withdrawn from contact with the tape which is rewound onto the main reel at a rate about twice its normal speed. The tape is driven entirely by the reel, not by the driving capstan, in this operation, hence its speed is not closely controlled and its acceleration is quite slow. The tape can therefore be rewound more quickly by a fast rewind than by a search operation (searching for the beginning of the tape) only if a traversal of more than about 35 feet of tape is involved.

Only one of the four operations (record, playback, search, rewind) can take place on any one transport at one time, although certain combinations can occur simultaneously on different transports. Movement of the tape must always cease at the end of one operation before another can be commenced on it.

Information can be recorded or played back only when the tape is moving at the proper speed. The transports on SILLIAC drive the tape at a speed of 75 inches per second. Although the tape is started and stopped very quickly by the driving capstans (the loops between the capstans and reels allow for the time lag in starting and stopping the reels) a certain amount of tape will still pass under the read/write head, during stops and starts, at a speed too slow for recording or playback. The length of tape involved in each case is approximately 0.9 inches.

Hence at any place on the tape where we may wish to stop the tape there must be an unused section of 1.8 inches in length. The tape will stop with the head in the middle of this section, allowing a distance of 0.9 inches for acceleration and deceleration in either direction. To minimise wastage of tape it is therefore necessary to arrange for there to be as few stopping places as possible. To achieve this we group the information recorded on the tape into blocks, each block consisting of a number of words, and we place the gaps in which the tape may be stopped only between blocks. The gaps are therefore called inter-block gaps.

Each end of a block is indicated by a special word called the block number word. The other words in the block are the information words. A block may contain any number of information

words, from a minimum of zero up to a maximum of about 1300. (The reason for the upper limit is described in section 12.8.) When we speak of a "block of n words" we mean n information words, or a total of $n+2$ words if the block number words are included.

The purpose of the block number word is to indicate the position of the end of the block and to carry a number, called the block number which identifies the block. The programmer is free to allot any block number to each block, the allowed range of values being from 0 to 4095, inclusive. (The block number uses 12 binary digits.) However, for ease of programming, it is customary to number the blocks serially along the tape. The block number is the only information identified by the search operation. The search operation is therefore used to position the tape at the end of a block carrying a selected block number.

When playing back or recording a block, irrespective of the motion of the tape, the block number word which marks the beginning of the block is called the block beginning word, while the one at the end of the block is called the block end word.

Each information word is a representation of a normal SILLIAC word, i.e. it contains 40 binary digits (or "bits"). As the tape is used purely as a "backing store", i.e. as an extension to the main store, there is no reason why we should convert the information to a different number system, as we do,

for example, when transferring information to and from paper tape. However to facilitate the internal mechanisms of the tape control and to provide an automatic error correcting code, the words on the tape are recoded (automatically) into a different form. Each 40 bit word is encoded into a total of 112 bits for recording on tape. When played back from the tape it is automatically decoded again into the normal 40 bits. The programmer need never be aware of the existence of this automatic coding feature of the tape although he benefits from it in that it makes the tape store very much more reliable than it would otherwise be.

The tape moves at 75 inches per second, and each word occupies 0.075 of an inch length of the tape. Hence, if the words are closely packed along the tape they will be recorded or played back at a rate of one thousand words per second, or one millisecond per word. However the words do not have to be packed close together. At the time the information is recorded the successive words can be transferred from the SILLIAC to the tape control unit at a rate equal to 1000 words per second, or any slower rate. If the rate is slower the words will be spaced apart on the tape, the space between the words being automatically erased. (Although very large gaps may be left between words within a block, the tape still cannot be stopped between these words. It can only be stopped between blocks.)

If SILLIAC produces the words at a rate higher than 1000 words per second it will have to wait before the tape control unit is ready to accept each word; thus the effective rate will be 1000 words per second.

There is an upper limit on the length of a block which limits, not the number of words, but the length of tape occupied by the block. This length is 100 inches. Hence, if the words are recorded at the maximum rate, the largest number that can be recorded in any one block is about 1300. If the words are recorded at a slower rate this number is proportionately reduced.

During playback the words will become available from the tape at the same rate[⊠] as that at which they were recorded. SILLIAC must be ready to accept each word as it becomes available. If SILLIAC is not ready in time the tape cannot wait, so the word will be lost. (This produces a hang-up condition in the tape control which prevents any other record or playback operation being commenced until it is reset.) If SILLIAC is ready too soon it will wait until the word becomes available.

The normal mode of operation in recording and playback is to have SILLIAC waiting upon the tape;

⊠ Approximately the same rate; the tape speed may vary by a few percent above and below the nominal speed of 75 inches per second.

this is essential in playback and ensures maximum density of recording in recording. However, for most efficient use of the SILLIAC, the time wasted in so waiting should, of course, be minimised.

The problem of efficient use of a magnetic tape store arises from the fact that it is a serial access store. As there is only a single read/write head associated with each tape, the tape must be scanned by this head to locate any required item on the tape, and the time taken by this scanning operation is a function of the distance that must be travelled. Hence, in planning a program that uses magnetic tape, consideration must be given to the relationship between the sequence of items as recorded along the tape and the sequence in which they are required to be played back.

The most efficient use of the tape backing store (in terms of time) is obtained when the tape is used in a serial manner, i.e. whenever the tape is in motion it is being used for recording or playback. However, this is frequently not possible, and we often use the tape as though it were a random access device. The search operation is autonomous, i.e. once initiated it proceeds independently of the SILLIAC which can proceed with other instructions, including record and playback instructions as long as they use a different tape to the one undergoing the search. If we are able to specify the required block number and initiate

the search sufficiently early to enable the operation to be completed before the time when the block is actually required, then the use of the tape as a random access store does not involve any time wastage.

12.2. TIMING AND SPACING ON THE TAPE

The speed of the tape is 75 inches per second, and the time taken to record one word (including the minimum interword gap) is one millisecond. The closest possible packing of words in a block is therefore 1,000 per 75 inches, or $13 \frac{1}{3}$ words per inch. The programmer may, if he so wishes, record words at a lower rate than 1000 per second. In that case each word is still recorded in the same time, but longer gaps of blank tape are left between words.

The arrangement of words in a block on the tape is as shown in Figure 12.1 (not to scale):

The process of recording a block automatically erases all information that may previously have been recorded on the section of tape lying between the starting and stopping points.

The starting and stopping distances are both approximately 0.9 inch, and the corresponding starting and stopping times are approximately 15 milliseconds, and 20 milliseconds respectively.

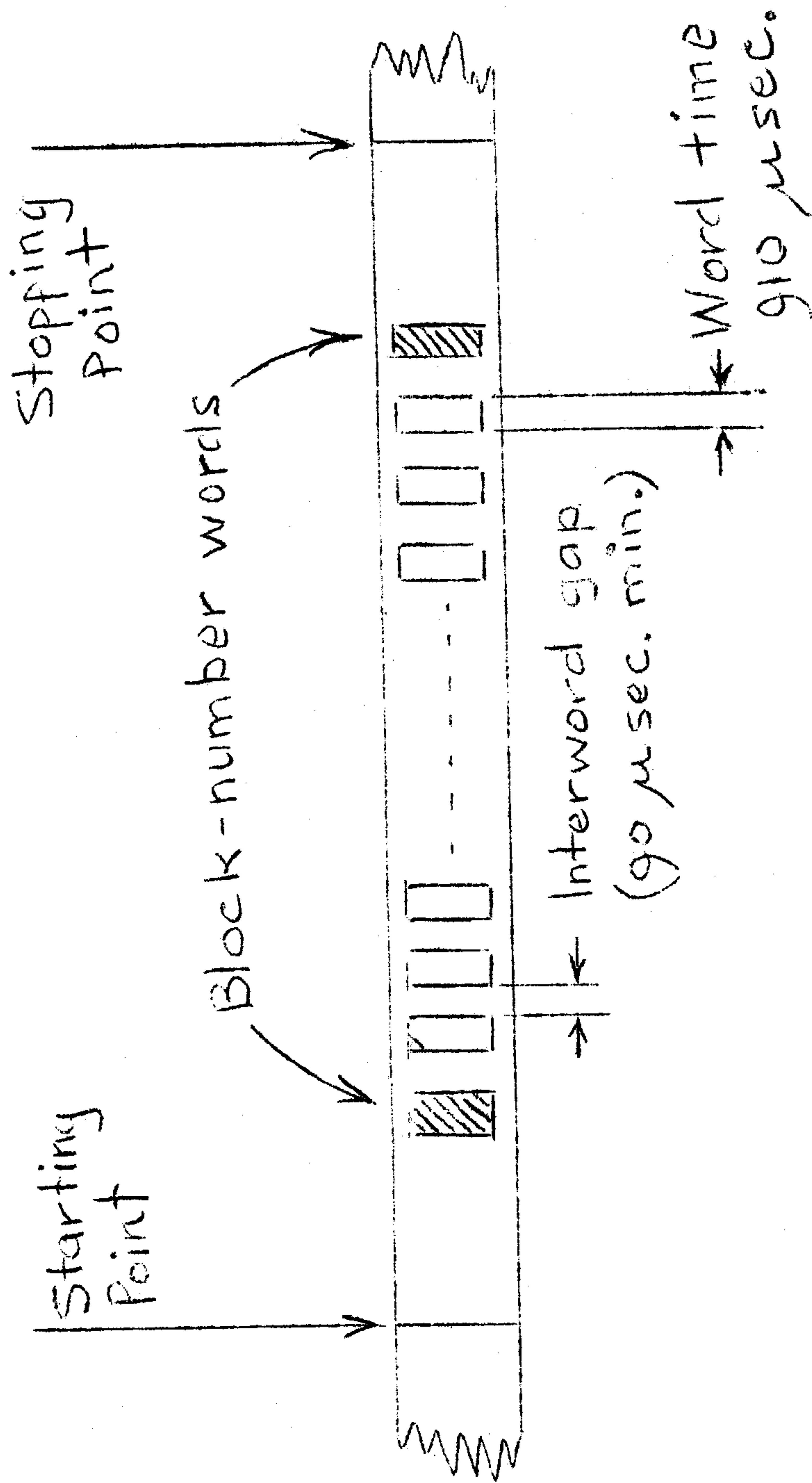


FIGURE 12-1

The starting time includes the time to operate the starting mechanism as well as the time to accelerate the tape and traverse the starting distance. Similarly the stopping time includes the time to operate the stopping mechanism, as well as the time to decelerate the tape, and an additional time for the equipment to reset itself ready to commence another operation. The starting and stopping distances and times are considerably longer than the minimum values of which the tape-handling equipment is capable, but they have been deliberately set at the higher figures for reasons that will be explained later. (See section 12.13). The distance between two adjacent blocks is the sum of the starting and stopping distances, namely 1.8 inch. This is the length of tape that would be traversed in 24 milliseconds at normal tape speed.

It should be noted that the starting time of 15 milliseconds is necessarily added to the time taken for recording or playing back the words of a block. The minimum time for a block of n words is therefore 16 plus n milliseconds (allowing 1 extra millisecond for the first block number word). The stopping time of 20 milliseconds is not necessarily wasted, since the SILLIAC is free to proceed with its program during this period. If, however, the SILLIAC attempts to record or play back another block within less than 20 milliseconds

after the last record, or playback operation, then it will be kept waiting until the full 20 milliseconds has elapsed. After this, the starting time for the next block is still the full 15 milliseconds.

12.3. DETAILS OF THE WORD ON TAPE.

To the programmer, a word on the tape is simply 40 bits as elsewhere in the SILLIAC. There is really no need for him to know the manner in which the forty bits are recorded. It will, however, be easier to appreciate the reasons behind the particular form chosen for block-number words, if the general form of all words is first explained.

Every word consists of sixteen characters, where the term character refers to a row of seven bits, one in each of seven tracks across the tape. The first and last of the sixteen are special marker characters labelling the beginning and end of words. They have significance only to the control circuits of the equipment, and never appear anywhere in numerical form. The remaining fourteen characters each consist of a redundant encoding of three binary digits. Since $14 \times 3 = 42$, while the word length required is 40 bits, two bits are wasted. The redundant code used is a single-error-correcting code so that the correct three bits will be obtained on playback provided that not more than one of the seven bits in a character has been corrupted.

In order to simplify the search process, block-number words are constructed so as to be symmetrical

with regard to direction of tape motion. The block number is a twelve-bit integer at the right-hand end of the word. If the characters be numbered 1,2 14, the block number occupies characters 11,12,13 and 14. To achieve the desired symmetry the same four characters are recorded in positions 4, 3, 2, 1 respectively.

As an example consider the block number 668 (decimal) which is 1234 octal. The full 40 bit word in this case would be as shown in Fig.12.2.

Octal	4	3	2	1	0	0	0	0	0	0	0	1	2	3	4
Binary	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0
Sexa-	8	J	1	0	0	0	0	0	0	0	0	2	9	N	
decimal															

Figure 12.2

two deleted bits

A given block number word may be either a block-beginning, or a block-end, depending on the direction of the tape motion. Two different special characters (called A-characters and B-characters) are used at the two ends of block-number words, and the equipment is thereby enabled to tell the difference between a block-beginning and a block-end. A given block-number word is recognized as a beginning, or an end, according as an A-character or a B-character is first encountered.

12.4. STORAGE CAPACITY

The maximum length of tape that can be used on a reel is 2400 feet. The number of words that can be recorded on this length depends, of course, on the word spacing, and on the block length. In any case

one should never attempt to calculate the capacity of a tape too exactly. Slight variations of tape speed, or of tape length, from nominal may send such a calculation astray, and lead to the inconvenience of running out of tape.

A conservative formula for the length of tape required for q blocks, having a mean length of p words per block, at closest spacing is:

$$\text{tape required} = \frac{q(p+26)}{150} \text{ feet}$$

Suppose, for example, one considers 100-word blocks. Then the above formula gives, for the number of blocks that one can be sure will fit on to a 2400 ft. reel, $q = 2,857$. So that the tape capacity under these conditions is 285,700 words.

The time required to pass over the full length of a 2400 feet reel with a search operation, i.e. at a speed of 75 inches per second without stops, is about $6\frac{1}{2}$ minutes. The time required to playback all the information on this length of tape, if it occupied 4,000 blocks would be 44 seconds greater. The fast rewind for a full reel is $3\frac{1}{2}$ minutes.

12.5. CONTROL OF THE TAPE

The processes of recording blocks on the tape, and of playing them back, are controlled by programs held in the main store of the SILLIAC. Such programs normally contain short loops of orders, and such loops

are traversed once for each word to be recorded or played back, as the case may be. The magnetic tape, once started, proceeds at a fixed speed of 75 inches per second until stopped at the end of a block. Because of this it is necessary, when writing program loops for recording or playing back, to give consideration to questions of relative timing of the magnetic tape equipment, and the SILLIAC proper. These matters will be discussed in more detail in following paragraphs.

For the purpose of handling the magnetic tape the SILLIAC order code contains the following orders:

- 8L Record Block Number
- 84 Record
- 94 Playback
- 96 Unlock
- 9L Search
- 86 Rewind

For a full statement of the details of operation of each of these orders the reader is referred to the Order Code dictionary in Chapter 3. The following descriptions are intended to amplify the statements there made, and to make clearer the manner in which the orders are intended to be used.

The tape transports are controlled by two units, called the "record/playback control" and the "search/rewind control". Each control unit is able to control only one operation at one time. Thus, if a record or playback operation is in progress on one tape transport no record or playback operations can be started on any

other transports, but a search operation can be commenced (unless one is already in progress). Similarly, if a search operation is in progress, a second one cannot be started, but a record or playback operation can be. If the record or playback operation is on the same tape transport as the search operation, then SILLIAC will hang upon the instruction (8L or 94) which starts the process until the search is completed, and will then proceed normally.

The fast rewind operation is a special one in that the search/rewind control is required only to initiate the operation. Once started, a fast rewind is carried out automatically by the tape transport itself. Hence, once a fast rewind operation has been initiated, another one, or a search operation, can be commenced; but a fast rewind cannot be started while a search operation is in progress. Fast rewinds can be carried out simultaneously on any number of the four tape transports coupled to the computer at one time. A fast rewind of any one of the five transports can also be initiated by manual operation.

The most complex joint activity of the four tapes coupled to the computer is to have one engaged in either record or playback, a second undergoing a search operation, and the other two being rewound.

12.6. ADDRESS DIGITS IN TAPE ORDERS

For extra flexibility in use the tape equipment has been designed so that recording and playback may take place with the tape moving in either direction. It must, of course, be remembered that if a block is recorded in one direction, and then played back in the other, the words will be received back from the tape in the reverse order from that in which they were recorded. It has, of course, been arranged that the order of digits within a word is unaffected by direction of tape motion. When direction of tape motion needs to be specified (8L, 94, 9L orders) the most significant address digit is used. A zero in this position specifies forward, and a one specifies reverse motion of the tape.

Of the five tape transports, any four (or less) can be coupled to SILLIAC at one time. Selection of the units to be coupled is made by manual plugging. (See section 12.15.) The coupled transports are referred to by the computer instructions as units 0, 1, 2, and 3. The number assigned to each transport is determined by manual selection. When one transport only is connected it is customary to assign the number 0 to it, if two are connected they are usually 0 and 1, and if three, they are 0, 1, and 2. It is physically impossible to assign the one number to two transports simultaneously.

Selection of a particular tape unit is made by the 2^6 and 2^7 digits of the address part of those

instructions (8L, 94, 9L, 86) in which such a selection is required. These two digits carry the unit number, i.e. 0, 1, 2, or 3.

Transfer of words between the SILLIAC and the tape is carried out via the Q register, and the process of shifting is employed in making the transfer. The number of shifts required is five. In orders which cause this transfer to be carried out (84, 94), the least significant six digits of the address must therefore contain the integer 5 (000101).

These uses of the several address digits are summarised in Figure 12.3.

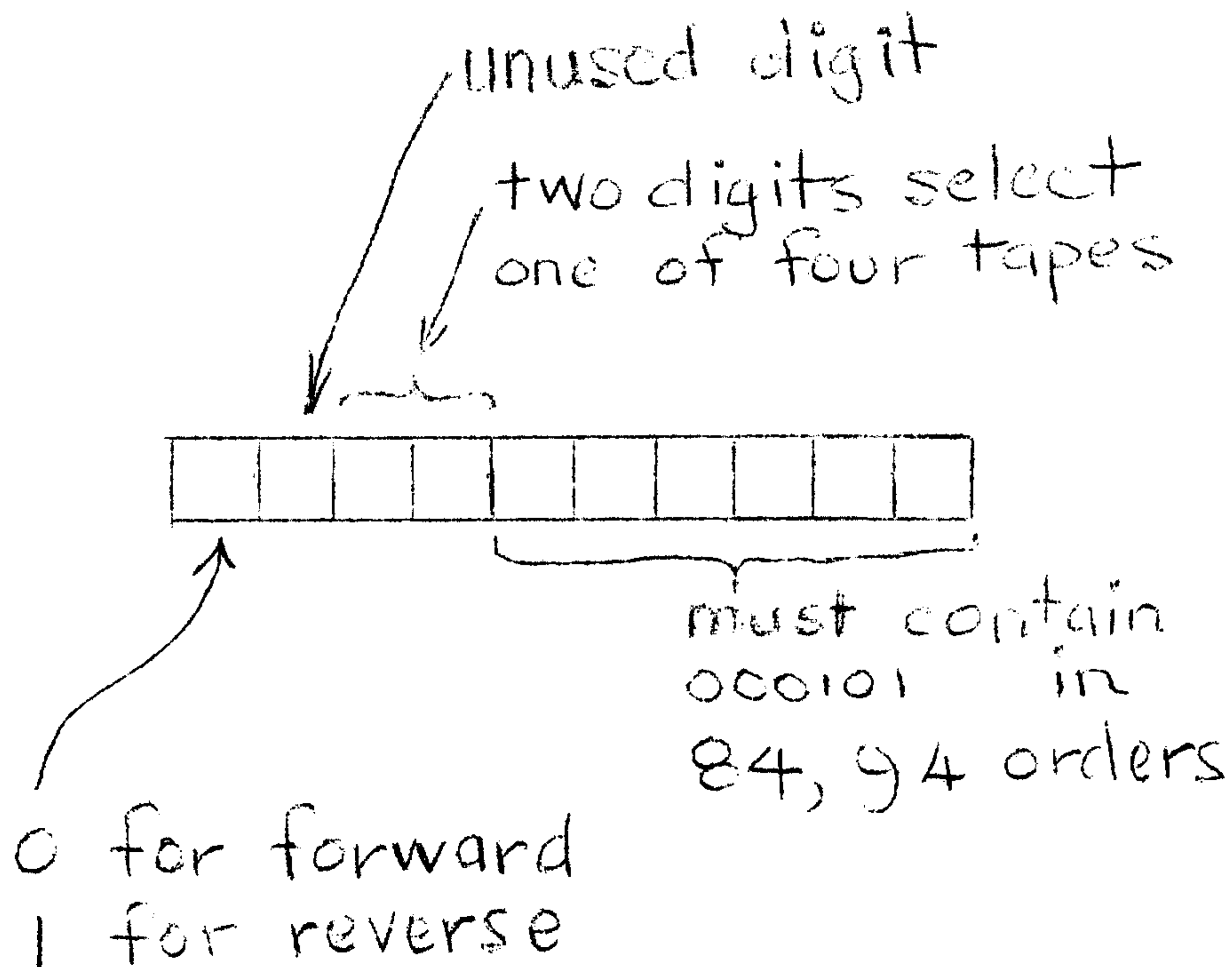


FIGURE 12.3

12.7. THE RECORDING PROCESS

In order to identify the individual blocks of words on the tape each block is allocated a Block Number by the programmer. Twelve binary digits are available for the block number, which must therefore be in the range 0 - 4095. The block number is contained in special block number words which are recorded one at each end of the block. These words are caused to be recorded by the Record Block Number order, 8L.

In order to reduce the number of separate orders that must be included in the order code and the number of orders required in a program loop for recording, the 8L order has been designed as a multi-functional order. It not only causes recording of block numbers, but also serves to start up the desired tape in the desired direction and at the end of the block, to stop the tape and cause control to be transferred out of the record loop. In order to carry enough information to perform these functions it is necessary that the 8L order occupy a full word. In D.O.I. form it is punched as 8L m F 00 n F, where m is a number chosen to select the required tape and direction of motion, and n is the desired block number.

A recording loop in a program will normally contain just one 8L order and one 84 order. When the loop is entered, and the 8L order is executed for the first time (the accumulator contents must be positive, or the machine will stop) the selected

tape is started up in the selected direction. When the tape has started, and has traversed the start - distance, recording of the block number word is initiated; simultaneously the SILLIAC is sent on to the left-hand order of the next pair. In the circumstances just described, the operating time of the 8L order is 15 milliseconds. This is the starting time previously mentioned.

When the tape has been started in the above manner, the tape equipment will be ready to receive a Record order (84) shortly before completion of the recording of the block number word. The limiting time is 900 microseconds. If the total operation time for the SILLIAC orders that intervene between the 8L order and the following 84 order is less than 900 microseconds, then the corresponding words on the tape will be at their closest spacing (90 microseconds interword gap). If, on the other hand, the total operation time for the intervening orders is greater than 900 microseconds, then the excess over 900 will add on to the interword spacing. The same limiting figure of 900 microseconds applies between successive 84 orders, and, at the end of the block, between the final 84 order and the final 8L order.

Each time the recording loop of orders is traversed, during the recording of the block, the 8L order will be encountered. The loop must contain a counting, or similar process, which ensures that the contents of the accumulator are

positive on the first and last occasions the 8L order is to be executed, and negative for all intervening occasions. On these intervening occasions, the 8L order is skipped over, and the SILLIAC goes on to the left-hand order of the next pair. The operation time for the 8L order under these circumstances is 37 microseconds. When, finally, the 8L order is executed with positive accumulator contents, the recording of the final block-number word is initiated, and simultaneously the SILLIAC is sent on to the right-hand order of the next pair; this must take the machine out of the record loop.

The following short open routine shows a simple example of a recording loop:

0	27 3L 3L 5L From 4	Start recording
1	50 q+1F 84 5F	Select word from store. Record it on tape.
2	F5 5L 40 5L	Increment counter.
3	8L 64xF From 0 00 bF	Record block number b on tape x in forward direction.
4	22 L 2----	Repeat loop. Exit after recording end of block word.
5	LL 4095F LL (4096-p)F By 0,2.	Counter.

Figure 12.4.

This routine would record the contents of storage locations $q-p+1$ to q inclusive, as a block of p words in the forward direction, on tape unit number x , with block number b . Record routines used in practice will in general be more elaborate than this simple example. Further examples will be found in section 12.20.

12.8. THE LIMIT ON BLOCK LENGTH

The reason for an upper limit on block-length, previously mentioned, will now be apparent. Suppose that a recording operation has been started, but is not stopped by a further execution of an 8L order with positive accumulator. This might happen either because of a program fault or a machine failure of some kind. Then, in the absence of some safeguard to deal with this possibility, the tape would simply keep running to the end of the reel, perhaps erasing many blocks of information in the process.

The upper limit of 100 inches for the length of a block has therefore been set. Shortly after this length of tape has run through, the tape will be stopped by the operation of a timing circuit built into the equipment. Once the tape has been stopped by the expiry of this time limit, the equipment cannot be operated further until it has been manually reset. (There is no limit on the duration of a search operation.)

12.9. THE PLAYBACK PROCESS

The playback process, like the recording process, is controlled by a program held in the main store of the SILLIAC, and such a program will normally contain a loop of orders that is traversed once per word of a block played back. The loop will normally contain just one playback order (94). The first time, after entering the loop, that the 94 order is executed, the selected tape is started up in the selected direction (both selections being made by the address of the 94 order), and the SILLIAC is kept waiting while the tape advances over the starting distance, past the block-number word, and on to the end of the first word of the block. At this stage, the digits of the first word have been assembled in a buffer register in the magnetic tape equipment, and this word is immediately copied into the Q register. A 5-place logical right-shift of the accumulator contents occurs during this process. SILLIAC goes on to the next order as soon as the copying into Q is complete.

The total operation time for the 94 order under the above described circumstances is nominally 17 milliseconds. This may vary by plus or minus a few milliseconds since the point from which the tape starts, i.e. the point at which it stopped after the previous operation, is subject to some variation.

When, after the tape has been started in the above manner, the 94 order is again executed in subsequent traverses of the playback loop of orders, it simply causes a word to be copied into the Q register as soon as it has been fully read from the tape. In order that this process should proceed correctly, it is necessary that execution of the 94 order should commence before the tape has advanced to the end of the word concerned. In order to ensure that failure to meet this requirement will not pass unnoticed, the equipment has been designed so that the SILLIAC will stop if a 94 order comes too late.

It is inevitable that the timing of playback operations cannot be specified as accurately as for recording operations. The timing during recording is determined by the recording circuits, which have a fairly high degree of stability; but the timing during playback is determined by the speed of the tape, which can vary over a few percent from nominal speed. Because of this, the total operation time of the orders intervening between the playback orders should not exceed 90 percent of the nominally available time.

For close-packed words on the tape the operation time between playback orders should not exceed 810 microseconds. For words that are not close-packed, the operation time between playback orders should not exceed 90 percent of the operation time between recording orders when the block concerned was being recorded.

In the simple case discussed above, it was supposed that the playback routine contained only one playback order, which served both to start up the required tape in the required direction, and subsequently to continue the process by causing successive words to be copied into Q. In a more elaborate routine one 94 order may be used to start the process and one or more separate 94 orders used to carry it on. In this case only the first order needs to contain the digits necessary to select the required tape and direction of motion. The subsequent 94 orders will automatically refer to the tape that is already running, irrespective of address digits. Of course the integer five must still appear in the right-hand six address bits, in order to specify the five shifts that are necessary to copy a word from the tape into the Q register.

The reason for employing five shifts in 84 5F and 94 5F orders will not be discussed in detail here. Briefly it may be said that the tape equipment contains a 40-bit buffer register, into which words to be recorded must be copied from Q ready for recording, and into which a word from the tape is assembled during playback, ready for copying into Q. Copying between Q and the buffer is carried out 8 bits at a time during each of five shifts, and therefore takes five shift times, or about 80 microseconds. This accounts for most of the minimum 90 microsecond interword

gap mentioned earlier. This is a small price to pay for the engineering simplicity of the arrangement used, as compared with attempting to copy all 40 bits in parallel in one operation. (Compare also the use of shifting in the processes of input and output, from and to paper tape.)

12.10. TERMINATION OF PLAYBACK

The programmer has a choice of ways of terminating a playback process, but whatever he does, the tape, once started by a 94 order will always run to the end of the block, and then stop.

If so desired, the playback loop may include a counting, or equivalent, process which causes control to be transferred out of the loop when sufficient words have been played back. This sufficient number of words may be the whole block, including the block-number word at the end, or any fewer number of words. If the playback of a block is deliberately stopped before playing back the final block-number word,[⊠] then the condition created is indistinguishable, so far as the equipment is concerned, from that in which the desired execution of a playback order is delayed or prevented by a program or machine fault. As has already been mentioned such a condition will cause the equipment to enter a locked-up state in which it cannot perform any further playback or record

⊠ Note that the 94 order does not play back the block beginning word, but it does play back the block end word.

operations (or any search or rewind on the tape concerned).

It is necessary then that the programmer should have available an order which will reset the equipment for further use after it has been placed in this locked state by deliberate abandonment of playback. The order which will perform this resetting is the Unlock order (96).

If SILLIAC attempts to execute a 96 order when playback has not in fact been abandoned, the machine will stop. A corollary of this is that delay will result if the 96 order follows too closely after the preceding 94 order, since playback abandonment is not registered until the tape reaches the end of the next word following the last one copied into Q by a 94 order. Not until then can the execution of a 96 order begin. Apart from any delay due to this cause the execution time of the 96 order may be taken to be 37 microseconds.

When it is desired to play back a whole block, including the final block-number word, there is no need to include, within the playback loop, any means for determining when the end of the block has been reached. The equipment itself detects the end of the block. After a 94 order which plays back the final block-end word, the SILLIAC goes on to the left-hand order of the next pair; after one of the preceding words, the SILLIAC goes on to the next order. Provided that the 94 order is itself a left-hand order, this difference may be used to transfer control out of the playback loop at the end of the block.

A word of warning may be issued here concerning a difficulty which may arise at the end of a block having wide-spaced words, when it is being played back by a lengthy playback loop which takes advantage of the wide spacing. During the course of the block it is immaterial whether the time interval associated with processing each word is considered to precede or to follow the word, but it does make a difference at the ends. When recording, the normal case would be for any time interval associated with processing each word to occur before it is recorded. In the absence of special measures, there would be no corresponding interval preceding the block-number word at the end, and this would then be close-spaced to the previous word. Trouble would then, of course, arise if the playback loop were such as to require wide spacing at this point.

12.11. THE SEARCH PROCESS.

It will always be desirable to write programs using magnetic tape so that no motion of the tape, other than that associated with recording or playback, is required. It will, however, seldom be possible fully to achieve this ideal, and frequently it will be necessary to move the tape to a new position, after completing one record or playback, ready for the next record or playback. The operation of Search has been provided to carry out this relocation of the tape to a specified position, with the least possible waste of time.

The search order, 9L, constitutes an instruction to the magnetic tape equipment to start up the specified tape in the specified direction, to proceed until finding a block with the specified block number, and to stop at the far end of this block.

The search operation compares the block number in both block beginning and block end words with the specified number. It stops the tape at the end of any block containing the correct number in the block number word at either end, but the operation comes to a normal conclusion only if both the block number words were correct. If the specified block number is found at only one end of a block a fault condition is set up, as described in section 12.16.

Note that the search operation always moves the tape right past the block being searched for, stopping in the interblock gap at the far end of this block. For this reason it is often more efficient to search for block $b-1$ or $b+1$ to prepare for an operation upon block b .

The program must supply the information as to the direction in which the tape must move to find the required block, since the equipment itself keeps no record of the position of the tape.

In order to contain all the required information the 9L order must occupy a full word. It takes the same form as the 8L order, i.e. in D.O.I. form, 9L mF 00 bF, where m is a number chosen to select

the required tape and direction of motion and b is the number of the required block.

The SILLIAC is not kept waiting while the search is performed. It is sent on to the next order as soon as the tape equipment has registered the information which block number is to be sought in which direction on which tape. The operation time for the search order is 55 microseconds.

While a search is in progress it is, of course, not possible for any other operation to be started on the tape concerned. Any attempt to do so will simply result in the SILLIAC being held up waiting for completion of the search to be signalled. In order to estimate the time required to complete the search the following formula may be used:

$$\text{search time} = \frac{26q + p_1 + p_2 + \dots + p_q}{1000} \text{ seconds}$$

where q is the number of blocks over which the tape must travel, and p_1 , p_2 , etc., are the numbers of words in these several blocks (not including the block number words).

Only one search control unit is provided. It is therefore possible to search on only one tape at a time. If the SILLIAC attempts to execute a 9L order while a preceding search is still in progress, it will stop and wait for this search to be completed, and then proceed normally.

The form of the block-number word has already been described. By using this arrangement, with the block number itself recorded in duplicate at the two ends of the word, the search process is simplified by being rendered independent of the direction of tape motion. Specifically, the process involves scanning the tape until the occurrence of a block number word is indicated by one of the special marker characters which identify these words. The next four characters are then read, and their content compared with the block number being sought. Thus one, or the other, of the two block numbers within the word will be read, depending on the direction of motion. The one read is always the first one encountered.

If all is well with the recording, the two block numbers within the same word will, of course, be the same. If, however, the recording is in some way faulty, it is quite possible that a particular block may be correctly located when searching in one direction but not when searching in the other direction.

12.12. REWIND.

A rewind instruction, 86, causes the selected tape transport to rewind its tape back to the beginning at the maximum possible speed, which is almost double normal working speed. A full reel of 2400 ft. which takes $6\frac{1}{2}$ minutes to run through at normal speed, can be rewound in $3\frac{1}{2}$ minutes.

The speed is not accurately controlled, and increases during the course of the rewind, so that only an approximate speed of 12 feet per second can be quoted for the purpose of estimating the time to rewind shorter lengths of tape.

The rewind operation is initiated by the same section of the equipment as handles the search operation. It is therefore not possible to start a rewind while a search is in progress. The rewind, however, goes on automatically in the tape transport, once it has been started. It ties up the search/rewind control only for about 10 milliseconds immediately following execution of the 86 order. The SILLIAC, therefore, does not have to wait for a rewind to be completed before going on with a search, or another rewind, on another tape unit.

As in the case of the search order the operation time of the 86 order is merely the time required for registration of the rewind instruction by the tape equipment. This amounts to only 37 microseconds. Following this, the above mentioned interval of 10 milliseconds must elapse before the search/rewind equipment can be used to initiate an operation on another tape.

After running each program using magnetic tape, the tapes may have to be replaced by fresh tapes for the next run, or the same tapes may be used again. In either case the tapes must be rewound. This

operation can be performed under manual control, or by rewind orders at the start of the fresh program, but time will be saved if programs contain, towards the end, the necessary 86 orders to rewind each tape as soon as the program has finished with it.

In some programs it may be necessary to reset a tape back to the beginning quite frequently during the course of the run. In such cases the programmer should consider whether the operation is best performed by a rewind, or by a search for the first block on the tape. If a substantial length of tape, or more than one tape, has to be rewound, the rewind process will in general be preferable. But if only a short length of one tape is involved, the search process may be faster. The operation times for the 9L and 86 orders are not significantly different, but another important consideration is the fact that a search will stop the tape right at the first block, whereas a rewind will cause the tape to be driven several feet further, until it is stopped by an automatic limit stop. The result of this is that the starting time for a recording or playback operation, following a rewind of the tape, is about 1.5 seconds instead of the normal 15 milliseconds. Furthermore, the rewinding takes about a second to get under way. Because of these features, a search will take up less time if the length of tape to be wound back is less than about 35 feet. For longer lengths a rewind will take up less time than a search.

12.13. RE-RECORDING OF BLOCKS

The tape system has been designed to allow any block on a recorded tape to be re-recorded with fresh information, by appropriately positioning the tape, e.g. by a search operation and carrying out a recording operation in the ordinary way. It is necessary however that certain precautions be observed if it is desired to make use of this feature.

The speed of the tape, and the position at which a tape stops after an operation on a particular block, are both subject to slight variation. The equipment has been set up to leave fairly long inter-block gaps to allow for these variations.[ⓧ] The important point to be noted, however, is that if a block is frequently re-recorded, and if that block itself is used as a reference for positioning the tape for each recording operation, then the block may drift along the tape until it begins to get too close to an adjacent block to allow of correct operation.

It follows that any block which is to be re-recorded should have adjacent to it, on at least one side, a block which is not at any time re-recorded. This adjacent block can then be

ⓧ The magnetic tape store could have been designed for maximum speed by reducing the start and stop times to the minimum imposed by the transport itself, but this would have made the re-recording of any but very short blocks a risky procedure. A small loss in speed has therefore been sacrificed for a gain in flexibility.

used as a reference for positioning the tape for each re-recording operation. If it is desired to have freedom to re-record any one of a long string of blocks, this can be handled by arranging, when the tape is first recorded, to place between every pair of blocks a special marker block containing no words other than the block-number words. Marker blocks for this purpose would never themselves be re-recorded and therefore will remain as fixed positions of reference on the tape. The tape can then always be positioned correctly by searching for one of the marker blocks. This solution has been adopted in a library tape-handling routine (Code Y1).

In the above discussion it has been tacitly assumed that when a block is re-recorded, the new block has exactly the same number of words as the old one. Clearly it could not have more words than the original, or it would not fit into the available space between the adjacent blocks. It is permissible however for the new block to have fewer words than the original. In that case part of the old block may be left unerased. This will cause no trouble provided that one does not attempt to play back through this unerased part of the old block, which can safely be passed over only under the control of a search operation.

12.14. END-OF-TAPE SENSING

It may be said, quite generally, of any magnetic tape store for a digital computer, that two separate end-of-tape sensing devices must be provided. One of these is in the nature of a safety device, a last-resort limit stop, which, if normal operation fails in some way to stop the tape, will take over and bring the tape to a stop before the end is literally dragged off the unwinding reel. The SILLIAC tape units have such limit stops at both ends of the tape. As well as acting as safety stops, they also enter into normal operation to some extent. In particular they are employed to stop the tape at the end of a rewinding operation.

In addition to the limit stops it is necessary to have some means whereby warning may be given that the limit stop is being closely approached. For the SILLIAC tape-end warning system, the tapes are marked at each end over a length of 100 inches measured from the final limit-stop points. When the tape is moving away from either end, recording is not allowed to commence until the tape advances beyond the marked region. When moving towards either end, any recording that may be in progress when the tape enters the marked region will be allowed to continue, but any attempt to record another block in the same direction will result in the SILLIAC stopping on the 8L order.

A program which records data onto a sequence of two or more tapes can either depend upon this end-sensing device to indicate the end of each tape, or it can keep a running record of the length of tape used (using the formula given in section 12.4.) If the latter method is adopted the program can automatically rewind the full tape and switch its attention to a different tape unit by modification of the addresses of the relevant instructions. It is a good plan to identify in some way the last block on each tape. This could be done, for example, by the recording of an additional word in every block; this word being zero in all blocks but the last, while in the last block it carries the reel number of the next tape (see section 12.18). Such an identification of the last block simplifies subsequent playback operations on this sequence of tapes.

If the programmer depends upon the automatic limit stop the computer will hang-up on an 8L order as described earlier, and it is then necessary for the operator to re-start the program. This could be done by bootstrapping in a sexadecimal word from paper tape which transfers control to the routine which performs the operations required when changing tapes. If the appropriate control transfer order has been placed in location zero by the program the same effect can be achieved more simply by a restart-at-zero operation. (White switch centre, press clear counter button, white switch down.)

The action carried out by the program on being restarted in this way could be to re-record the last block successfully recorded, adding an additional identifying word, to rewind this tape, and to modify instructions so as to refer to the tape unit on which the next tape has been placed.

The action of the warning device on playback is similar to that on recording. That is to say, the SILLIAC will stop on a 94 order which attempts to start a playback towards the end of the tape when the tape is already less than 100 inches from the end of the reel. The operator can then re-start the program as described above for the recording operation.

In the case of the search operation the end-warning device has no significance at all. If a tape reaches the end with a search still in progress, it will be stopped by the limit stop. If required, the operator can change to another tape and continue the search.

12.15. MANUAL CONTROLS

The manual controls provided consist of:

- (a) Two 3-position key switches, one for Record/Playback, and one for Search/Rewind. The three positions of each switch are

RESET: Up, with spring return to centre.

HOLD: Centre

RUN: Down

- (b) Five plugs, one for each tape transport, and four jacks, one for each of the four selection lines from SILLIAC. Any of the five plugs can be inserted into any of the four jacks.

In addition, each of the five transports has the following controls:

- (c) REMOTE/LOCAL switch
- (d) STANDBY/OFF/ON switch
- (e) REVERSE/STOP/FORWARD switch
- (f) REWIND push button
- (g) RECORD DISABLED/PERMITTED switch

Controls (d), (e) and (f) are located behind a small door at the top of the tape transport. The REMOTE/LOCAL switch is automatically set to LOCAL by opening this door, and to REMOTE by closing the door. In order to be operated by SILLIAC the transport must be switched to REMOTE.

There are other controls associated with turning the equipment, as a whole, on and off, and with certain testing operations. These are not discussed here. The operations of putting a reel on to the transport, and of removing a reel from the transport, are also not discussed here. These operations will be performed by the regular machine operators. Individual machine users will not be permitted to handle magnetic tapes.

12.16. AVAILABILITY OF THE TAPE STORE TO SILLIAC -
EQUIPMENT HANG-UPS.

In order to write efficient programs employing the magnetic tape it is necessary to have some appreciation of the organisation of the tape store, and of the manner in which its organisation affects the availability of the equipment to the SILLIAC. The equipment consists essentially of three sections:

- I. The Record/Playback Control Unit.
- II. The Search/Rewind Control Unit.
- III. The several tape transports.

At the commencement of any operation the control unit concerned attempts to select the desired transport. If, for any reason, the transport concerned is not available, the control unit will stop (hang up) and wait for it to become available. Reasons for non-availability of a tape transport include:

- (a) No transport is plugged into the selection channel concerned;
- (b) The transport is not turned on, or is switched to local, or has no tape in it, or is rewinding;
- (c) Tape in the transport has already reached the end for the direction of motion ordered;
- (d) The transport is already selected by the other control unit.

Cases (a) and (b) may be due simply to an operator's error. In this case the operator can switch the control unit concerned from RUN to HOLD, repair his omission, and switch back to RUN. The

operation should then proceed normally. Case (c) may arise because of a program error, or may arise normally when using a tape file that extends over more than one reel of tape. Case (d) will be a frequent normal occurrence, and in this case the delay is only temporary. For example, the SILLIAC may be kept waiting while search control completes a search on a particular transport and releases it, before playback control can select this transport. In good programming such delays will be kept to a minimum.

Once one of the two control units has been called into operation by a SILLIAC instruction, it becomes unavailable for any other operation until the first has been brought to a normal conclusion. Various reasons why a normal conclusion may not be reached are discussed below, and in any of these cases the control unit will hang up until reset by the operator. Any transport which may have been selected by the control unit concerned will, in these circumstances, also remain unavailable, so that a hang-up of type (d) above may occur in the other control unit if this attempts to use the same transport.

Circumstances which may cause the record/playback control to hang up include the following:-

- (e) The transport is not available for one of the reasons listed under (a), (b), (c), or (d);
- (f) An attempt is being made to record on a tape for which recording has been disabled.

- (g) A playback order has failed to appear in time. If this occurs as a result of a program fault, the SILLIAC will hang up on the playback order when it does finally arrive. If it occurs as a result of deliberate playback abandonment the hang-up will be cleared by execution of an unlock order.
- (h) If, after commencing to play back, a block-end is encountered on the tape before a block beginning, the tape will stop and the record/playback control will hang up. The SILLIAC will also hang up on the playback order which attempted to start the operation. Manual resetting is required in this case.
- (i) It is possible for a block-end to be partially recognised, enough to cause the tape to be stopped, but not fully recognised so as to complete the operation normally. In this case the record/playback control hangs up, and the SILLIAC also, if it is attempting to play back the block-end word. If playback has been abandoned before this condition arises, the SILLIAC will not be caused to stop until the next time it attempts to use some part of the equipment that is hung up. Manual resetting is required.

Circumstances which may cause the search/rewind control to hang up include the following:-

- (j) As (e) above.

- (k) A block is found with the sought-for block number at one end, but not the other. Manual resetting is required.
- (l) The tape arrives at the end without having found the required block. If it is desired to carry on the search on another tape the control must be switched to HOLD while changing the tape or plugging in another transport.

Specific attention may be drawn here to a difference between record/playback on the one hand and search/rewind on the other. When initiating a record or playback, SILLIAC cannot go on to the next order until the required transport has been selected, started up, and the operation is fully under way. With search/rewind, the SILLIAC may have to wait for the control unit to become available, but does not have to wait for the required transport to become available. The control simply registers the instruction, sends the SILLIAC straight on to the next order, and then carries out the instruction as soon as the transport becomes available, if it is not already so.

There is also a difference between search and rewind. The control must monitor the progress of a search until it is completed, and is therefore not available for anything else. A rewind, however, only has to be started, a process which takes only 10 milliseconds following the moment when the transport becomes available. From then

on only the transport remains unavailable for the duration of the rewind, so that the control becomes available to initiate another rewind, or to carry on a search.

When a resetting operation releases equipment that has been hung up, any other operations that have been waiting for the equipment to become available will immediately proceed. Suppose, for example, that the record/playback control has hung up, that subsequently a search instruction requiring the same transport has been registered by search control, and that subsequently again the SILLIAC has attempted operation of a 94 order to initiate playback of another block again on the same transport. SILLIAC will hang upon the 94 order. Then if record/playback is reset, the search will immediately proceed, and in due course the playback order in SILLIAC's order register will also proceed. If it is not desired that the registered instructions proceed in this way, then they must be cleared out prior to the resetting operation.

An instruction registered by search/rewind control can be cleared out by resetting that control. Instructions standing in the SILLIAC's order register can be removed either by raising the white switch to the up position (preparatory to a bootstrap start) or by raising the white switch to centre, and pressing the Counter Clear button (preparatory to a restart-at-zero). If it is

desired merely to hold up the standing instructions temporarily, rather than to clear them out altogether, this can be done by placing the tape equipment control switches in the HOLD position. This produces an effect identical with that of any required transport being for the time being unavailable.

12.17. DIAGNOSING A COMMON PROGRAMMING ERROR

Early experience with using the tape store has shown that a common error consists of attempting to play back beyond the end of the series of blocks so far recorded. The symptoms observed when this occurs will depend on what, if anything, was recorded on the tape during its previous use. Because of this the symptoms will not necessarily be the same each time the erroneous program is run. Unless forewarned of this, a person code-checking may be misled into thinking that an equipment fault is involved.

If the fault under discussion occurs with a new tape, or one that has been freshly subjected to bulk erasure, there will be nothing recorded beyond the file written by the current program. In this case, attempting to play back in the blank tape region will result in the equipment hanging up on the expiry of the built in time-limit, after a little more than 100 inches of tape have run through.

In most cases the tape in use will have old information recorded on it. If one then attempts

to play back beyond the end of the currently recorded series of blocks, the most likely event is to encounter a partially over-written old block. A block-end will then be read before a block beginning, and the equipment will hang up in the manner previously described for these circumstances. It is, of course, possible that a complete block, rather than a partially over-written block, will be found and played back, without any fault becoming obvious until a later stage of the run.

12.18. FILE TAPES AND WORKING TAPES

The various uses of a magnetic tape may be grouped into two classes:

- (1) A tape reel may be used as a medium for permanent or semi-permanent storage of information which is to be carried over from one computer run to another. Such a tape is called a file tape. Examples of such use are:-
 - (a) A large amount of data may need to be processed several times by the computer. The data is recorded on a magnetic tape reel (or perhaps several reels if the volume requires it) which is mounted on the tape transport for each run. Such data may be statistical data, the accumulated results of many experiments, the output of a computer Monte Carlo calculation, etc.

- (b) A lengthy mathematical calculation with low volume input and final output may require storage for very large amounts of intermediate results and will, of course, use magnetic tape for this purpose. If the duration of the calculation is greater than 30 minutes it must be designed so that it can be taken off the computer to be re-started later. It is then necessary that its magnetic tape reel be dismounted and stored away, to be re-mounted on the transport when the computation is resumed.
- (c) A program which consists of a very large number of instructions may be recorded permanently on a magnetic tape. When such a program is to be used the tape reel is mounted on a transport, and a short program, punched on paper tape, is used to bring the first stage of the main program into SILLIAC. The same magnetic tape, or another on a different transport, may also be used to hold intermediate results produced by the program.

Before a tape reel can be used in this way it must be reserved for exclusive use in conjunction with a particular project.

(2) A tape reel may be used purely for temporary storage of intermediate results during a computation which is sufficiently short to be completed in a single machine run. The information left on such a tape at the conclusion of the computation is of no value and can be immediately overwritten by the next program. Such tapes are called working tapes.

All magnetic tapes, file or working, are kept at all times within the computing laboratory. When not mounted on a transport they are stored in an air-conditioned cabinet. The tapes are sensitive to sudden changes in temperature and humidity and must be kept scrupulously clean. The handling and care of magnetic tape reels must be left at all times to the laboratory staff.

A tape reel to be used as a file tape must, of course, be reserved for use solely with the program(s) of a particular programmer. Either full reels of 2400 feet, or half reels of 1200 feet, may be reserved. As a file tape may contain a very large amount of information which, if lost, would be expensive to replace, it is important that every precaution be taken against the inadvertent destruction (by overwriting) of such information.

Essential information on a file tape may be overwritten either through (1) a programming error (which may remain undetected, perhaps for a long time, until a particular combination of parameters arises), (2) an operator error (placing the wrong

tape reel on the transport, assigning the wrong numbers to the transports used, placing the wrong paper tape in the reader, etc.) or (3) a machine malfunction.

Causes (1) and (3) can be very effectively guarded against by arranging the program so that no additional information is recorded on the file tape. Then the transport on which this tape is mounted can be switched to "record disable". It is then impossible for any programming or machine error to erase or overwrite any information on that tape. If the information on the tape is to be up-dated it should all be transferred to a second tape reel (on another transport) the alterations being made during the transfer. The original tape should be retained until the information has been successfully played back from the second tape, thereby guaranteeing that it has been correctly recorded. Preferably the original tape should be retained until the data has been transferred, in the normal course of events, to a third tape reel. The second tape, in its turn, is retained until the third has become redundant, the information having been transferred to a fourth (which may well be the first reel again). If this course is followed there will always be one tape, not in active use, containing the information required to repeat the last machine run should the tape in active use be accidentally damaged. In some programs the recovery of the original information from paper tape may be preferable to such a cyclic use of magnetic tapes.

Failures of type (2) can be effectively guarded against by suitably programmed checks. It is the responsibility of each programmer to see that his program contains sufficient checks to prevent any important information being lost through the mounting of a tape on the wrong transport or the use of a wrong paper tape. But it is also necessary that his tape be protected against corruption through being inadvertently used in conjunction with a different program.

To provide an automatic safety check, additional to the normal care exercised by the machine operators in selecting file tapes from the storage cabinet, every file tape is identified by having a special two word block recorded as the first block on the tape. This block, which carries block number zero, is called the tape label.

The first word of the label is the same for all file tapes, and serves to distinguish them from working tapes. Working tapes carry no label for it would not be possible to preserve one against the vagaries of programs in course of de-bugging. The file tape identifying word is:

In sexadecimal: 433L9 NF8L3

In DOI form: 43 1018F NF 2291F

When the sexadecimal form of this word is printed in letter shift it reads

REELONFILE

The second word of the label distinguishes the file tapes from each other. It contains the project number in the first sexadecimal characters. The remainder of the word is used to distinguish the different tapes of the one project. For example, the first tape reserved for project 288/1 would carry as the second word of the label the following sexadecimal characters

288-100001

Every program which uses one or more file tapes must contain a routine which checks each tape used by the program to see that it is the correct tape. This check, which involves playing back both words of the label, must be performed on each tape before its use by the program at each machine run. An example of a routine to perform this check is given in section 12.21 as Figure 12.10.

Every program which uses one (or more) working tapes must contain a routine which checks that the tape(s) on the selected transport(s) is indeed a working tape. This involves playing back the first word of the tape and checking that it is not "REELONFILE". An example of such a routine is given in section 12.21 as Figure 12.11.

It should be noted that these checks can usually be performed as interludes which are carried out during the reading in of the program and hence occupy no storage space while the program is running. However this is not so

in the case that a tape must be changed during the operation of the program.

12.19. CHECKING FOR TAPE ERRORS

The automatic error correcting code which is "built in" to the magnetic tape store guarantees a very high reliability in the recording and playback operations. When a "working" tape is used to provide additional working storage space for a calculation of relatively short duration (say, less than 20 minutes) there is little to be gained by attempting to increase the reliability very far above that of the rest of the computer. Hence, in such cases, additional checks on the accuracy of the tape store are not warranted. However, with working tapes used with programs of longer duration, and with all file tapes the reliability of the tape is exceptionally important.

The simplest check that can be made on the correct recording and playback of information is a sum check. In its simplest form this is obtained by forming the sum of all the information words in the block as it is recorded on the tape, then recording minus this sum as an additional word before the final block number word. In the playback operation the sum of all information words (including the check sum word) is formed as they are read, and, after the complete block has been played back, the sum is checked to see that it is zero.

If the sum is not zero a second attempt can be made to play the block back. The program should stop the computer before such a second attempt is made so that the operator can note that an error has occurred. The conventional stop order in such a case is 8F or 9F. The operator will restart the machine by bypassing the stop order with the white switch. If two or three such restarts fail to read the block correctly it will be assumed that this block is illegible, either through a machine malfunction during the recording process, or because of a physical defect of the tape itself (such as a piece of oxide coating missing).

As a check on the correct recording of a block a playback operation can be performed immediately after the block is recorded. In this operation it is possible to compare each word with the copy kept in the main store, but it is sufficient to provide the same check as on normal playback, i.e. to check that the sum of words, as played back, is zero. If this test fails the computer should be stopped (on 8F or 9F) before a second attempt is made to record the block. If such a stop occurs twice in succession the maintenance engineer may be called in to inspect the tape for physical defects.

In programs in which a very high reliability of the tape store is not essential the check sum can be omitted. However, as there is, in most applications, ample time to form the check sum

during the record and playback operations, and the check sum word on tape represents a very small addition to the amount of stored data, the use of this check is customary in all cases. But the immediate playback of each block after recording, which does waste considerable time, can be omitted.

The complete omission of a check sum is justifiable only in those cases in which available space is so restricted in the main store that the additional instructions required in the recording and playback routines cannot be accommodated.

12.20. SOME TYPICAL MAGNETIC TAPE ROUTINES

The record and playback routines described here transfer a block of information between the main store and the magnetic tape, for this is, by far, the most common way in which the tape is used.

The shortest known record loop is as shown in Figure 12.4 (on page 12.21). This routine records the contents of storage locations $q-p+1$ to q , inclusive, as a block of p words on tape number x , moving in the forward direction, with block number b .

Note that the 27 3L instruction ensures that the "record block number" instruction 8L 64xF 00 bF is first obeyed with the accumulator positive. This causes tape x to be started in motion in the forward direction and the first block number word to be recorded. The counter which must initially have the value $-p \cdot 2^{-39}$, as shown in Figure 12.4, is

used both as an index for the instruction 50 q+1F, and also as a counter. After p words have been recorded the counter becomes positive and the 8L instruction then records the block end word and causes control to jump to the right hand side of word 4.

The duration of the instructions between successive 84 instructions is 330 microseconds, hence the words are recorded close packed on the tape at the maximum rate of 1 millisecond per word.

The efficient use of tape often requires that a block should be recorded with the tape moving in one direction, and subsequently played back with the tape moving in the opposite direction. Unless some caution is used this could lead to an inversion of the sequence of the words within the block. The simplest way to prevent this is to impose a consistent rule to determine the sequence of the words on the tape. The convention normally followed is that when recording or playing back in the forward direction the words are selected in a sequence corresponding to successively increasing addresses in the main store, whereas with the tape moving in the backward direction we select words with successively decreasing addresses.

To modify the routine of Figure 12.4 to record the block with the tape moving in the backward direction we would replace the 3L 5L instruction by 3- 5L, the 50 q+1F by 50 q-pF and would add 512 to the left hand address of the 8L instruction.

Figure 12.5 is a simple example of a record loop which computes a check sum and records it as an additional word at the end of the block. This routine records exactly the same block as that of Figure 12.4, with the difference that the additional check sum word makes the block $p+1$ words long. The check sum word is equal to minus the sum of the other words in the block, hence the sum of all words is made to be zero. Note that, as the last word of the block is a special word, it is necessary to provide a test for the end of the block in addition to the one provided automatically by the 8L instruction. This is done in word 2. At this time the check sum is in Q; hence transferring control to word 6 with a positive accumulator causes the check sum to be recorded and the block to be terminated. The five left shifts incurred in the 84 5F instruction do not affect the sign of the accumulator at any time, for its content is always a small integer.

The duration of this loop is 685 microseconds, hence the block is recorded at the maximum rate of 1 millisecond per word.

An alternative way of recording a check sum is to compute the sum in a separate operation, storing it at the end of the block in the main store. Then a simple record loop, as in Figure 12.4, will record the block complete with the check sum. But such a method is slower (for the summing is not done while the computer is waiting for the tape between words)

0	-- -- 41 10L	Clear check sum
1	<u>26 7L</u> 50 10L From 8.	Record first block number. Put check sum in Q.
2	F5 9L 32 6L	Test for end of block.
3	3L 9L 50 q+2F	Select word from main store.
4	L5 10L -0 F	Subtract it from check sum.
5	40 10L F5 9L	
6	40 9L 84 5F From 2.	Increment index word. Record contents of Q.
7	8L 64xF From 1. 00 bF	Record block number word and test sign of A.
8	22 1L 2..... 	Exit after recording last block number word.
9	LL 4095F LL (4096-p-1)F	Index word.
10	(00 F 00 F)	Check sum.

Figure 12.5.

and will be found to require additional storage space for the instructions needed.

The routine of Figure 12.5 can be modified for recording in the backward direction in a similar way to that of Figure 12.4. However in the present case we meet the difficulty that reversal of direction of tape travel will result in the check sum being recorded at the opposite end of the block. Hence a block recorded in the forward direction will differ from one recorded in the backward direction, thus leading to difficulties with the playback operation.

This situation can be met by the expedient of recording a "check sum word" at both ends of the block. The first check sum word recorded will be zero, the last one will be minus the sum of the other words. In playback, one word from each end of the block is discarded and the sum of all words, including the discarded ones, is checked to see that it is zero. Using this technique a block may be played back in either direction without regard to the direction in which it was recorded.

Figure 12.6 is a routine which records the contents of locations $q-p+1$ to q , inclusive, on tape x in the forward direction as a block of $p+2$ words carrying block number b . The operation of this routine is as follows. Starting at word 0 we clear the check sum to zero, then start the tape moving and record the first block number word by obeying the 8L instruction with a positive accumulator. Then in

word 1 we put the check sum, which is zero, in the Q register and go to word 6, for the accumulator is still zero. The left hand instruction in word 6 places itself in the accumulator. The purpose of this is to make the A_5 digit unity so that after the 84 5F instruction has been obeyed the sign of A will be negative. Thus the zero word in Q is recorded on the tape and, as A is negative, we go back to word 1, then through the 36 instruction to word 2. Now follows a series of p cycles in which successive words are extracted from the store, subtracted from the check sum, and recorded. The counter in word 9 then becomes zero and, with a zero accumulator, the 32 6L instruction in word 3 is obeyed. This results in the check sum, which is in Q, being recorded and, as A is positive, the block is terminated and control goes to the right hand side of word 8.

The duration of this loop is 703 microseconds, hence the block is recorded at the maximum rate.

To record the block with the tape moving in the reverse direction we change 3L 9L to 3- 9L, 50 q+1F to 50 q-pF and 8L 64xF to 8L 512+64xF.

0	41 10L <u>27 7L</u>		Clear check sum. Record first block number.
1	50 10L 36 6L	From 8.	Put check sum in Q. Test for beginning of block.
2	F5 9L 40 9L		Increment index word.
3	32 6L 3L 9L		Test for end of block.
4	50 q+1F L5 10L		Select word from main store.
5	-0 F 40 10L		Subtract it from check sum.
6	L5 6L 84 5F	From 1. From 3.	Make $A_5 = 1$. Record ⁵ contents of Q.
7	8L 64xF 00 bF	From 0.	Record block number word and test sign of A.
8	26 1L <u>2.....</u>		Exit after recording last block number word.
9	LL 4095F LL (4096-p-1)F		Index word.
10	(00 F 00 F)		Check sum.

Figure 12.6

The shortest playback loop is as shown in Fig. 12.7.

0	26 2L	Start here.
	NO(q)F	By 1. From 2. Store in selected store location.
1	F5 L	
	42 L	Increment store location.
2	94 64x+5F	Playback word.
	22 L	
3		From 2. Exit after reading last block number word.

Figure 12.7.

This routine reads a block from tape unit x in the forward direction into consecutive storage locations from q onwards. There is no need to specify the length of the block as the loop is automatically terminated by the transfer of control to the left hand instruction in word 3 when the end of the block is reached. The duration of this loop is 240 microseconds, hence it is able to playback a block recorded at the maximum density.

If the block is to be played back in the backward direction the address of the NO instruction will need to be decreased in each cycle. This will require the loop to be a word longer. (Note that the 94 order must be a left hand order.) Of course, the initial address of the NO instruction will then be the address of the upper end of the block in the main store.

Playing back a block with a single check sum word is quite simple if there is sufficient space in the main store for us to place the check sum word there at the end of the block. In this case the routine of Figure 12.8 is suitable.

0	41 8L <u>26 4L</u>	From 7.	Clear check sum. Commence playback.
1	L5 8L -4 F	From 4.	Add word to check sum.
2	40 8L NO (q)F By 3.		Store word in selected location.
3	F5 2L 42 2L		Increment storage location.
4	94 64x+5F <u>26 1L</u>	From 0.	Playback word.
5	L3 8L 36.....	From 4.	Exit if check sum is zero.
6	94 512+64x+5F 96 F		"Backspace" over block.
7	9F F <u>26 L</u>		Stop. Repeat.
8	(00 F 00 F)		Check sum.

Figure 12.8.

The playback loop differs from that of Figure 12.7 only in the inclusion of three additional instructions for forming the sum of all words played back. In word 5 we test to see if this sum is zero. If it is not we "backspace" the tape over the block

just read, in preparation to reading it again in the same direction, and then stop on a 9F instruction.

The method used in Figure 12.8 to backspace over the block is to set the tape in motion with a 94 instruction then to obey an "unlock tape" instruction (96) so as to eliminate the need for any additional 94 orders. We could equally well have used the loop

```
6  94 512+64x+5F
   26 6L
```

to pass over the block. The technique we have used has, in some other applications, the advantage that it frees SILLIAC to carry on with other instructions as soon as the second word of the block has been read. In our case there is no real advantage for the time lost in stopping the computer for operator attention outweighs all other factors.

The routine of Figure 12.8 suffers from the disadvantage that it stores the check sum word in the main store. There may be applications in which this cannot be allowed, there being just enough space in the store for the original block, without the check sum. The routine of Figure 12.9, which reads a block with a sum word at each end, avoids this defect. The technique used there could equally well be applied to the simpler routine of Figure 12.8.

0	94 64x+5F NO 10L	From 9.	Read first sum word. Store it in 10L.
1	94 5F <u>26 4L</u>		Read first true word.
2	L5 11L 40 (q)F	From 6. By 3.	Store previous word in selected location.
3	F5 2L 42 2L		Increment storage location.
4	-5 F 40 11L	From 1.	Store word temporarily in 11L.
5	L4 10L 40 10L		Add word to check sum.
6	94 5F <u>26 2L</u>		Playback word into Q.
7	L3 10L 36.....	From 6.	Exit if check sum is zero.
8	94 512+64x+5F 96 F		Backspace over block.
9	9F F <u>26 L</u>		Stop. Repeat.
10	(00 F 00 F)		Check sum.
11	(00 F 00 F)		Temporary store.

Figure 12.9.

The duration of this loop is 575 microseconds, hence it is able to play back blocks which are recorded at the maximum density.

To modify the routine so that it can play back from a tape moving in the backward direction we must lengthen it by one word so that we can reduce the address of the 40 (q)F instruction in each cycle. Also the address of this instruction must be initially equal to the address of the upper end of the storage block in the main store, and the addresses of the 94 instructions in words 0 and 8 must be interchanged. Note that the addresses of the other two 94 instructions do not need to specify the tape number or direction, for they never initiate a motion of the tape.

12.21. ROUTINES FOR CHECKING TAPE LABELS

When a file tape is issued to a programmer it has two blocks recorded at the beginning of the tape. The first block is the tape "label". It has block number zero and contains two words. The first word consists of the ten sexadecimal characters "REELONFILE", the second contains the programmer's project number and a serial number assigned by him to distinguish this tape from others assigned to the same project. The tape label should never be overwritten or erased.

The second block carries block number 1, and consists entirely of a block beginning word and a block end word. This is a "marker block" of the

type used by CODE Y1 and serves to protect the label from overwriting by Y1 (and similar library routines). The label is protected against all operations of Y1 except the "prepare blank tape" operation which should, therefore, never be used with a file tape. The programmer may overwrite the marker block if it is of no use to him (i.e. if he is not using Y1).

Figure 12.10 is an example of an interlude which checks the tape on transport number x to see that it is the correct file tape. In this example the second word of the label is supposed to contain 288-100001 (i.e. reel number 1 of project 288/1).

0	80 40F 40 F		Read two sexadecimal words and store in 0 and 1.
1	80 40F 40 1F		
2	86 64xF 92 705F	From 6	Rewind tape x. Print Figure Shift.
3	94 64x+5F <u>22 6L</u>		Play back first word of tape x, go to 4 if block is empty.
4	92 129F 92 913F	From 3,10.	Print CR/LF " XXXXX (5 multiplication signs)
5	92 165F 96 F		Print 10 CR/LF's Stop.
6	<u>26 2L</u> <u>-5 F</u>	From 3.	
7	L0 F 40 2F		
8	L3 2F 32 10L		Test first word.

9	94 5F	From 13	
	26 9L		Finish off block.
10	26 4L		Print error signal.
	<u>94 5F</u>	From 8	
11	-5 F		
	L0 1F		
12	40 2F		
	L3 2F		Test second word.
13	36 14L		
	<u>26 9L</u>		
14	92 129F	From 13	Print CR/LF
	92 593F		" = = = = = (5 equal signs).
15	92 165F		" 10 CR/LF's.
	96 F		Unlock tape.
16	24 999F		Return to DOI.
	26 L		
	26 1N		
	REEL ON FILE	Test	
	288-1 00001	words.	

Figure 12.10

The SILLIAC output switch is turned to "printer" when the program is being read in. If this interlude finds that the tape label is correct it prints five equal signs and returns control to the DOI via a 24 order. This stops SILLIAC to enable the operator to switch the output to "punch" before proceeding with the program read-in. The magnetic tape is left located at the end of the label block.

If the label is found to be incorrect in either word, or if the tape has no label, the interlude prints five crosses (i.e. multiplication signs) and

SILLIAC stops on a 96 order ("unlock tape"). This order causes a hangup because playback has not been abandoned; in fact the order pair in word 9 has completed the playback of the label block. The 96 stop is the one conventionally used, in conjunction with a print-out, to indicate to the operator that the incorrect reel is on the transport. (It suggests to the operator that SILLIAC is trying to "abandon" the reel). The operator can now replace the tape reel then by-pass the 96 stop. The test will then be repeated.

A somewhat more complex program would obviously be required to test the tapes on several tape transports.

Figure 12.11 is an example of an interlude which checks the tape on transport number x to see that it is a working tape. This interlude simply checks to see that the first word on the tape is not "REELONFILE". The operating procedure for this interlude and the print-out it yields are the same as with that of Figure 12.10. The only difference is that a tape which passes the test is rewound before the interlude transfers control back to the DOI.

0	80	40F		Read "REELONFILE" and store it in 0.
	40	F		
1	86	64xF	From 11	Rewind tape x.
	92	705F		Print Figure Shift.
2	94	64x+5F		Play back first word of tape.
	22	5L		
3	92	129F	From 2,8	Print CR/LF
	92	593F		" = = = = .
4	92	165F		" 10 CR/LF's.
	86	64xF		Rewind tape x.
5	24	999F		Return to DOI.
	-5	F	From 2.	
6	10	F		
	40	1F		
7	96	F		Unlock tape.
	L3	1F		
8	36	9L		Test first word.
	26	3L		
9	92	129F	From 8.	Print CR/LF
	92	913F		" XXXXX
10	92	165F		" 10 CR/LF's.
	96	F		Stop.
11	26	1L		Repeat test when re-started.
	26	L		
	26	1N		
	REEL ON FILE			Test word.

Figure 12.11

12.22. MAGNETIC TAPE BOOTSTRAPS

We are sometimes faced with the following problem. What is the smallest number of instructions required initially in the main store to play back a program recorded in one, or more, blocks on a magnetic tape? The number of instructions required can be reduced to a minimum of two, or more reasonably three, by the use of the "bootstrap" technique, i.e. by having nearly all the instructions required stored on the tape and using the initial instructions in the store to bring in only the first pair.

If the tape is already located at the correct position (if not, a search instruction pair will place it there) the three instructions

```
94 64x+5F
NO  F
26  F
```

Figure 12.12.

will suffice to bring in the bootstrap routine of Figure 12.13. Note that the 26 F of Figure 12.12 can be omitted if the other two orders are in location 1023.

```
94 5F NO 1F
94 5F NO 2F
26  F 00  F
94 5F NO  F
F5 1F 40 1F
94 5F NO mF
```

Figure 12.13.

The bootstrap routine assembles itself into locations 0,

1 and 2 as shown in Figure 12.14,

0.	F5	1F	40	1F
1.	94	5F	NO	(m)F
2.	26	F	00	F

Figure 12.14

then proceeds to play back the remaining words of the block, placing them in successive locations from m+1 onwards. Playback will proceed until all locations from m+1 to 1023 are occupied. The next word goes into location 0 and is then immediately obeyed. This word will normally be the last word of the block and will consist of a 96 instruction, to eliminate the need to playback the block end word, and an instruction transferring control to the routine just read in.

If the block does not contain enough words to fill the storage locations from m+1 through 1023 to 0, the input routine of Figure 12.14 will play back a succession of blocks until the requisite number of words have been read. Each block end word will not be stored, but the contents of the corresponding store location will be left undisturbed. If used in this way it is necessary that the 94 instruction in word 1 (i.e. the 94 instruction in the sixth word of the block) specify the tape number in its address.

If it is desired that the block to be read in from magnetic tape be placed, not at the upper end of the store, (from m+1 to 1023), but at the lower end, the first few words on the tape should read:

```

94 5F NO 1F
94 5F NO 2F
94 5F 22 F
F5  F 42 F
94 5F 22 F
26 xF 00 F

```

where 26 xF is the instruction that will transfer control to the program after it has been read in. These instruction pairs will be read by the instructions of Figure 12.12 and will place themselves in locations 0, 1, 2 and 3 as follows:

```

0. 94 5F NO (4)F
1. F5  F 42  F
2. 94 5F 22  F
3. 26 xF 00  F

```

They will then play back the following words of the block and will place them in consecutive locations from 4 onwards. When the end of the block is reached the instruction in the left hand side of location 3 is automatically obeyed.

Either of the above two bootstrap routines can be used to play back a sum checking playback routine, such as the one of Figure 12.8 or 12.9, which would then bring in one or more other, longer, blocks under the control of a check sum.

A block can be read from magnetic tape into a completely empty store by executing a bootstrap start with the sexadecimal word

940y5 N0001

in the paper tape reader, where y is four times the

number of the tape transport concerned. Either of the above bootstrap routines can be brought in this way. In both cases the initial 94 5F NO 1F instruction pair should be omitted.

To bootstrap in the contents of a file tape immediately after it is placed on a transport is more complicated, for we must first check the tape label. A possible solution to this problem is to take the label checking routine of Figure 12.10 and change it so that, after a successful test, it ends with the instructions of Figure 12.12 instead of with 24 999F. This modified routine can then be punched on a paper tape together with a DOI, or preferably, a sexadecimal copy complete with bootstrap input can be made.

CHAPTER 13

THE PROGRAM LIBRARY

The problem of planning and carrying out a large-scale computation is often almost entirely a matter of organization in which the detailed parts of the computation may be subcomputations which are common to many problems. Consequently, it is of great importance for the efficient use of a large-scale computer to have readily available to the programmer means for carrying out these subcomputations. For example, it would be very wasteful if each programmer had to prepare his own routine for converting binary numbers in the machine to decimal numbers.

The program library is a collection of routines which have been prepared to make coding easier. They have all been machine tested and copies punched on tape are kept available for use. All are prepared for use with the Decimal Order Inputs (see Chapters 4 and 5).

The library programs may be divided into several categories:

- Input routines
- Output routines
- Functional routines
- Problem-solving routines
- Interpretive routines
- Checking routines

13.1 INPUT ROUTINES. These are programs for transferring data on punched tape to the machine. There are routines in the library which will input decimals or fractions, singly or in sequences.

13.2 OUTPUT ROUTINES. These routines convert the binary numbers in the machine to decimal fractions or integers and punch the results as determined by wide choices in parameters.

13.3 FUNCTIONAL ROUTINES. These are routines which evaluate functions for specified values of the arguments. Included in the library are functional routines for finding the following quantities:

- square root,
- cube root,
- integral root,
- sine,
- cosine,
- arctangent,
- logarithm,
- exponential,
- Legendre polynomial value.

13.4 PROBLEM-SOLVING ROUTINES. Problem-solving routines carry out more elaborate operations than functional routines. Those in the program library include routines for carrying out the following operations.

- Solving a set of first order differential equations,
- Integration,
- Solving a set of linear algebraic equations,
- Fitting a least squares line,
- Solving Laplace's equation,
- Minimizing a function of n variables,
- Interpolating,
- Inverse interpolating,
- Finding real roots of $f(x) = 0$,
- Multiplying matrices,
- Finding eigenvalues and eigenvectors,
- Solving Poisson's equation.

13.5 INTERPRETIVE ROUTINES. These routines (see section 4.6) carry out more elaborate "orders" in their own order code by using many SILLIAC orders. The library contains interpretive routines for the following operations:

Double precision arithmetic,
Floating decimal arithmetic,
Automatic coding,
Minimizing Boolean polynomials.

13.6 CHECKING ROUTINES. These are routines designed specifically to aid the programmer in obtaining information about his program. They are discussed in Chapter 8.

13.7 SUMMARY. A summary of the routine currently (15.12.58) available in the SILLIAC Library follows. Full write-ups of any of these routines are held in the "blue books" at the Laboratory, and copies of the write-ups of most of the routines may be obtained from the Program Librarian.

A1

Floating Decimal Arithmetic Routine

This interpretive routine (with its own order code) will carry out normal arithmetic operations (including input, output, multiplication and division) on numbers in the form $X \cdot 10^p$, using a floating decimal accumulator. X and p must lie in the ranges $1 > X \geq 1/10$, and $64 > p \geq -64$. This routine eliminates the need for scaling a program, but occupies 168 storage locations.

A2

Floating Decimal Tape with Standard Auxiliaries

This tape contains seven programs (viz. Floating Decimal, Number Conversion, Square Root, Sine, Exponential, Logarithm, Constant Listing Auxiliary) arranged for convenient use. Each, except the last, is addressed by means of an S parameter. The sum check routine is provided at the end of the tape to ensure correct input.

A3

Conversion from Floating Decimal Representation to Normal Machine Form

This closed subroutine, for use with floating point auxiliaries, will convert one half times the number N in the floating accumulator into standard binary form, placing the result in the AQ register, the binary point appearing between $(A)_{39}$ and $(Q)_1$. If $|N| < 10^{-10}$ zero will be placed in AQ and if $|N| > 10^9$, the portion in the first register of the floating accumulator will be placed unchanged in A.

A4

1.7 Precision Floating Binary Arithmetic with Floating Decimal Conversion

This interpretive routine should be used when an accuracy of up to 20 significant decimal places is required, and also the scaling of the problem justifies a floating point routine. Numbers are input and output in the form $X \cdot 10^p$ (where $-1 = X < 1$ and $|p| \leq 153$) but actually operated on in the form $(x_1 + 2^{-39}x_2)2^E$ with x_1 and x_2 and E stored in 2 consecutive memory locations. The normal operations including multiplication and division are provided for.

LABEL

TITLE and BRIEF DESCRIPTION

A5 Complex Number Arithmetic

An interpretive routine (with an order code almost identical to that of the floating decimal routine) which carries out arithmetic operations on numbers in the form $(A + iB) \times 10^P$. Two registers are used for the storage of each complex number.

Included in the routine are the auxiliaries:

- (a) replacing a number by its complex conjugate
 - (b) replacing a number by its modulus
 - (c) replacing a number by the principal value of its square root.
-

A6 Floating Decimal Routine and Auxiliaries

This tape contains the seven programs included in code A2, with the exception of the last (constant listing auxiliary), and in addition has an arc tangent auxiliary.

A7 Semi-Interpretive Floating Binary Arithmetic Routine

This semi-interpretive routine carries out the arithmetic operations of addition, subtraction, multiplication and division on floating binary numbers, normal fractions, and normal integers. It is designed for programs in which the amount of floating point computation is sufficient to warrant the use of a special routine, but speed requirements make a fully interpretive scheme undesirable.

A8 Interpretive Floating Point with Auxiliary for Easy Use

This program is a combination of the following library routines:

A1, A3, RA1, TA1, SA2, SA3, TA2, RA2, XA1. S parameters have been assigned to each of these routines and to available storage space. The programming required to use this routine is very simple and makes this program extremely useful.

A9 Interpretive Floating Point with Simplified Coding Scheme

This program is an improved version of A8, and should supersede it.

LABEL

TITLE and BRIEF DESCRIPTION

B2

Complex Number Operations

This is an interpretive sub-routine to perform on complex numbers the operations normally performed by the type 6,7 or L orders. The real part and imaginary part of each complex number must be stored in consecutive memory locations.

LABEL

TITLE and BRIEF DESCRIPTION

C1 Post Mortem Version of the Decimal Order Input

A discrepancy between the contents of the memory and the program tape may be detected by reading the program tape back into the memory with this routine instead of the D.O.I. at the beginning of the tape. The address of the discrepancy, the word on the tape and the word in the memory will be printed.

C3 Print Sets of Signed Fractions and then Decimal Locations

This is a post mortem which will print for checking specified sets of ten consecutive, signed, 12-place decimal fractions, preceded by their respective memory locations in decimal form (the presence of numbers which are zero is indicated by a blank line). A 2-digit decimal number n is read in from the tape, and the first set of ten will begin at the location whose address is $10n$.

C4 Print Sets of Decimal Integers and then Locations

This post mortem prints signed 12-place decimal integers and their decimal locations in the same manner as code C3 prints fractions.

C5 Print Sets of Order Pairs and their Location

This post mortem checking routine is similar to codes C3 and C4. The contents of the selected sets of ten consecutive memory locations are printed as order pairs preceded by their decimal locations.

C6 Address Search Routine

This routine will locate all the words in the store which, when considered as order pairs, have a given address p in either the left or right hand position or both. On being restarted it will then locate words with addresses $p-1$. It is used to discover why control went astray, or why a particular word at address p was maltreated.

C7,C8 Address Search Routine

These routines are used in exactly the same way as C6, but differ from it in that C7 looks only for transfer of control orders, and C8 looks only for store orders.

LABEL

TITLE and BRIEF DESCRIPTION

- C9 Print Sets of Floating Decimal Numbers and their
Decimal Locations
This post mortem is for use with routines A1 or A3, and is similar to routines C4 and C5. It will print specified sets of ten floating decimal numbers in the form, sign plus a 2-digit exponent.
-
- C10,C11 Print Details of Last Interpretive Order and Contents
of Floating Accumulator
These post mortem routines are employed when a program stops under control of an interpretive routine. C10 may be used with A1 or A4, and prints to 9 decimal places, while C11 may be used only with A4, and prints to 20 places.
-
- C12 Post Mortem Version of Sum Checking D.O.I. X12
This program is used with X12 as C1 is used with X1, to print out discrepancies between the store and the program tape.
-
- C13 Half-Precision Version of C3
Used in the same way as C3, this post mortem routine prints the contents of specified sets of store locations to five decimal places.
-

LABEL**TITLE and BRIEF DESCRIPTION**

D1 Check Point Code II

If part of a program is to be checked, special "blocking orders" are placed by this routine at points of the program specified by the programmer. These will cause a transfer of control to this routine, which then prints out the information specified by the programmer on a "specification tape". The information to be printed out may come from any location and the number of times information is required from any particular check point can be specified. A wide variety of forms of print-out is available.

D5 Iteration Counter

When a program is run under the control of this routine it will print out the number of times any specified conditional transfer of control order is used before there is a change of the sign which it tests. Thus it provides information as to how many times a cycle of orders is repeated.

D15 Control Transfer Check Routine

This routine is read into the store after the main program has been read in. D15 replaces the order pair at the required point by a blocking order pair specified on the tape. The program will continue from this pair under control of this routine. As each control transfer order is obeyed, it is entered in a list in a specified part of the store. When the store space allocated to this list is full, the older entries are overwritten by the new ones in a cyclic fashion. The list is printed out when the program stops, and by studying the printout, the course of the program in the last stages can be traced.

D7 Economic Check Point Routine

This diagnostic routine is similar to D1, but occupies much less store space (no storage beyond that normally occupied by the D.O.I. being used). The routine is not as versatile, and the specification tape must contain the information required when an order is blocked for every time that order is blocked.

LABEL

TITLE and BRIEF DESCRIPTION

E2 Integration by Simpson's Rule

This closed subroutine computes an approximation to $\int_0^1 f(x)dx$ by means of the formula,

$$\int_0^1 f(x)dx \approx 1/3n(f_0 + 4f_1 + 2f_2 + \dots + f_n)$$

where n is the number of intervals (even) and $f_i = f(i/n)$.

The subroutine is used in conjunction with a subroutine supplied by the programmer for computing $f(x)$ for a given x.

If an integral over a general range of integration is to be computed, the formula

$\int_a^b ydv = (b-a) \int_0^1 ydx$, where $v = (b-a)x+a$, may be used.

E3 Integration by Simpson's Rule (Function Values)

This closed routine performs the same operation as D2, but it is to be used with a subroutine supplied by the programmer for computing $f(x)$ for a given x. The f_i are not tabulated but are computed as required.

E4 Numerical Differentiation with Interpolation

This routine employs Neville's interpolation formula and its derivatives to compute $f(x)$ and $(h^r/r!)f^{(r)}(x)$ for given x and arbitrary r, in terms of known function and derivative values at equally spaced mesh points. The accuracy depends on the mesh size and value of x chosen.

E5 Quadrature by Gauss' Method

This is closed routine, using the approximation

$$\int_0^1 f(u) du \approx \sum_{i=1}^n a_i f(u_i)$$

to evaluate the integral.

Tapes are given with values of u_i and a_i for values of n from 2 to 16 inclusive.

The values $f(u_i)$ must be obtained by an auxiliary subroutine supplied by the programmer.

LABEL

TITLE and BRIEF DESCRIPTION

E6 Inverse Laplace and Mellin Integration

This closed sub-routine may be used to compute an integral of the following form.

$$f(m) = \frac{1}{2\pi i} \int_{b-i\infty}^{b+i\infty} e^{mz} Q(z) dz$$

where $Q(z)$ is a given function such that $Q(\bar{z}) = \overline{Q(z)}$, and such that the integral along the line $\operatorname{Re}(z) = b$ is the same as the integral with the same integrand taken along the parabola $y^2 = 4a(b-x)$

E7 Integration over a Single Interval

This routine computes an approximation to the integral

$$(1/w) \int_{A_p}^{A_p+w} f(x) dx$$

when $f(x)$ is tabulated for s values of x equally spaced at an interval w on each side of a_p .

LABEL

TITLE and BRIEF DESCRIPTION

F1 Solution of a System of Ordinary Differential Equations

This subroutine will solve any set of simultaneous differential equations which can be expressed in the form:

$$y'_0 = f_0(y_0, \dots, y_{n-1})$$

$$y'_1 = f_1(y_0, \dots, y_{n-1})$$

$$\vdots$$

$$y'_{n-1} = f_{n-1}(y_0, \dots, y_{n-1})$$

The coder must write an auxiliary closed subroutine to evaluate the functions $f_0, f_1, \dots, f_{n-1}$ from the given y_i .

Gill's modification of the Runge Kutta method is used.

F2 Solution of a system of Differential Equations by Milne's Iterative Method

This routine will solve n simultaneous first order equations or a set of equations reduced to this form, provided that in the reduced equations the first derivatives are expressed explicitly as functions of the variables. Milne's method uses two quadrature formulae, the first of which predicts a value of the function, the second of which then makes a correction.

F3 Integration of n Simultaneous Second Order Differential Equations with Initial Conditions Specified

This routine integrates n second order differential equations of the type,

$$y''_i = f_i(y_0, y_1, \dots, y_{n-1}, y'_0, y'_1, \dots, y'_{n-1})$$

$$i = 0, 1, \dots, n-1$$

An auxiliary routine, which evaluates $h^2 y''_i$ from y_i and $h y'_i$ must be called in a number of times equal to the number of iterations needed for convergence.

F5 Integration of a System of Ordering Differential Equations up to a Specified Value of One Variable

This closed subroutine will integrate a system on n simultaneous first order ordinary differential equations.

$$\begin{aligned} y_0' &= f_0(y_0, y_1, \dots, y_{n-1}) \\ y_1' &= f_1(y_0, y_1, \dots, y_{n-1}) \\ &\vdots \\ y_{n-1}' &= f_{n-1}(y_0, \dots, y_{n-1}). \end{aligned}$$

Integration is carried out by the Gill-Kutta method described in routine F1, from a set of initial values of the variables until one specified variable y_K reaches the value Y_K .

FA1 Second Order Linear Differential Equation with Two Point Boundary Conditions

A closed auxiliary for the floating decimal routine (A1) It will compute the solution of the differential equation $Ay'' + By' + Cy = D$ at the equally spaced points x_i ($i = 0, 1, \dots, n$) subject to the boundary conditions

$$\begin{aligned} Dy' + Ey &= G \Big|_{x=x_0} \quad \text{and} \\ Hy' + Ky &= M \Big|_{x=x_n}. \end{aligned}$$

A, B, C, D, E, F, G, H, K, M are all functions of the independent variable, and the first four are evaluated at the given x_i by an open subroutine supplied by the programmer (using interpretive orders).

FA2 Floating Decimal Solution of a System of Ordinary Differential Equations

This is another closed auxiliary to the floating decimal routine. It will solve n simultaneous first order simultaneous equations which are expressed in the form,

$$y_i' = f_i(y_0, y_1, \dots, y_{n-1}), \quad i=0, \dots, n-1.$$

The coder must write a closed subroutine which evaluates the f_i at the given y_i .

G1 Laplace's Equation - Liebmann Method

This code gives the solution $V(x,y)$, to the difference equation which is an approximation to Laplace's equation,

$$\partial^2 V / \partial x^2 + \partial^2 V / \partial y^2 = 0,$$

subject to given boundary values of V .

The routine uses an approximate solution supplied by the programmer at each interior point of a mesh, and improves upon it by taking the average of the function values at the four neighbouring points, the process being repeated until there is no appreciable change in the value of V .

G2 Poisson's Equation, Liebmann-Frankel Method

This code finds the solution, $V(x,y)$, of the difference equation which approximates to Poisson's equation,

$$\partial^2 V / \partial x^2 + \partial^2 V / \partial y^2 = \rho(x,y),$$

subject to given boundary values of V .

An initial approximation supplied by the programmer must first be made at each point of a square mesh, covering a region. This is successively improved upon by an iteration formula which converges more rapidly than that used with G1.

H1	<u>Inverse Interpolation, a Real Root of $f(x) = 0$</u> An initial interval (x_1^0, x_2^0) of the independent variable is supplied by the programmer, and $f(x_1^0)$ and $f(x_2^0)$ are computed by an auxiliary routine. If $f(x_1^0) \cdot f(x_2^0) \leq 0$, this routine will make a first approximation X^0 to a root by inverse linear interpolation. $f(X^0)$ is tested for sign. If it is positive, the reduced interval of the independent variable will be (x_1^0, X^0). If it is negative, the reduced interval will be (X^0, x_2^0). The process is repeated until either $f(X^n) = 0$ or $x_1^n - x_2^n \leq 2^{-30}$, where n gives the number of iterations.
H2	<u>A Search for the Real Roots of $f(x) = 0$</u> This is a routine which includes the previous one (code H1), the Decimal Fraction Print routine (code P2) and orders which examine a range, $x_0 < x \leq x_0 + nh$, of the independent variable x for roots of $f(x) = 0$. If $f(x) = 0$ has roots in the i-th interval, $x_0 + (i-1)h < x \leq x_0 + ih$, then no roots will be recorded if there is an even number, and one root will be recorded if there is an odd number.
H3	<u>Minimization of a Function of Two Variables</u> This routine will find a minimum, $f(x_0, y_0)$, of $f(x, y)$, and record the values x_0, y_0. It consists of an "inner loop" which continues to change x and y by amounts δ_x and δ_y respectively, calculated at the beginning of the loop, until no improvement is obtained. Then it computes a new δ_x and δ_y and repeats the process. This is the "outer loop". When there is no appreciable decrease in $f(x, y)$ after several "outer loops", an apparent minimum has been obtained. This is improved by the automatic reduction of the mesh size used in the calculation of δ_x and δ_y. The final accuracy depends on the final mesh-size.
H4	<u>Minimization of a Function of Four Variables</u> The method used for the minimization of $f(x, y, z, t)$ is similar to the one used above for $f(x, y)$.

LABEL

TITLE and BRIEF DESCRIPTION

H5	<u>Minimization of a Function of n Variables</u> The function $f(x_1, x_2, \dots, x_n)$ is minimized by a method similar to that used by Code H3 for the minimization of $f(x, y)$. $x_1, x_2, \dots, x_n$ are altered by increments $\delta x_1, \delta x_2, \dots, \delta x_n$ respectively to obtain decreases in the function value.
H6	<u>Minimization of a Function of n Variables by Treating One Variable at a Time</u> This routine examines $f(x_1, x_2, \dots, x_n)$ at $x_1 \pm \delta$ (the mesh size), by proceeding in the direction of decreasing f until f is minimized as a function of x_1. This is repeated for each x_i ($i = 2, \dots, n$) in turn. We then return to x_1, x_2, etc., until no further decrease in $f(x_1, x_2, \dots, x_n)$ is obtained. Further improvement is obtained by change of mesh size.
H7	<u>Minimization of a Badly Conditioned Function of Two Variables</u> This closed subroutine finds the minimum of a function of two variables $F(x, y)$ and the range of values of the independent variables over which the function differs from its minimal value by less than a specified amount. It then calculates the value of an arbitrary function $G(x, y)$ and the range of values of G over the region defined above.
HF1	<u>Zero of a Solution of a Differential Equation</u> A program parameter is selected, which defines the variable, y_i, which is to vanish; then the routine repeatedly enters Library Routine F1 (Solution of Ordinary Differential Equations), adjusting the size of the steps, until y_i becomes zero.

LABEL**TITLE and BRIEF DESCRIPTION**

I1

Interpolation

Neville's method of successive linear interpolation is used to find the value of a function, $f(x)$, at any point x , where $x_0 < x < x_n$, and $f(x_i)$ is tabulated at the $n+1$ equally spaced points x_i ($i=0, 1, \dots, n$).

LABEL

TITLE and BRIEF DESCRIPTION

J1 Roots of a Polynomial

This is a complete program which uses a third order iterative procedure to find all the roots of a polynomial $f(x) = \sum_{i=0}^n a_i x^{n-i} = 0$, where the a_i may be complex.

J2 Roots of a Polynomial

This is a complete program which uses a 1.8th order iterative procedure to find all the roots of a polynomial whose coefficients may be complex numbers. The duration is $0.082n^2 + 5.2n + 18$ seconds, where n is the degree of the polynomial.

LABEL

TITLE and BRIEF DESCRIPTION

K1 Best Least Squares Straight Line

This is an open routine which fits a straight line to a set of up to 1,000 pairs of points x_n, y_n . The coefficients a and b of the line $y = a + bx$ are printed out.

K2 Product Moment Correlations, Means, Standard Deviations, Variances and Covariances

This is a complete program which computes the matrix of correlation coefficients between each pair of a set of up to 38 variables. The means and standard deviations are also recorded in two parallel columns, and variances and covariances are computed and output.

K3 Least Squares

A polynomial of the form, $F(x) = 1/2 \sum_{s=0}^{n-1} a_s x^s$ is

fitted to N weighted experimental points, so that the sum of the deviations,

$$M = \sum_{i=0}^{N-1} (F(x_i) - f(x_i))^2 w(x_i)$$

is a minimum for arbitrary variations of the coefficients a_s . The $f(x_i)$ are the given experimental data.

K5 Autocorrelations

A set of up to 795 observations will be manipulated to give the autocorrelation coefficients for a specified lag; this may be specified in any one of four ways. The mean and variance will also be computed.

K9 Product Moment Correlations, Means, Standard Deviations, Variances & Covariances in Logarithmic Scale

This is a complete program which includes Code K2 (q.v.) and Library Routine S1 (Natural Logarithms) plus a short section to compute $\log_{10}$ from $\log_e$. This routine may be used when a linear relation between variables exists in a logarithmic scale only.

LABEL

TITLE and BRIEF DESCRIPTION

K10 Calculate Autocorrelations of a Time Series

From a set of fractions or integers, $x_1, x_2, \dots, x_n$
 $(0 \leq |x_i| \leq 10^4)$ this program will compute

$$R_k = \frac{\overline{x_i \cdot x_{i+k}} - \overline{x_i} \cdot \overline{x_{i+k}}}{\sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2 - \overline{x_i}^2}}^{1/2} \quad \left\{ \begin{array}{l} \text{for } k=0, 1, \dots, m' \\ i=1, 2, \dots, N-k \\ \text{and } M8 < N/4 \end{array} \right.$$

where $\overline{x_i} = \left(\frac{1}{n} \sum_{i=1}^n x_i^2 - \overline{x_i}^2 \right)^{1/2}$
 and $\overline{x_{i+k}} = \left(\frac{1}{n-k} \sum_{i=1}^{n-k} x_{i+k}^2 \right)^{1/2}$ etc.
 The x_i are the supplied data.

K11 Design of 2^N Factorial Experiments

This complete program enables the automatic determination of admissible treatment combinations in fractional replicates and block replicates of the

$2^N = 2^A \times 2^B \times \dots$ series of factorial experiments. If required, a random permutation of the experimental objects is also determined by Monte Carlo method.

K13 Quartimax Orthogonal Rotation of Factors

This program obtains the quartimax solution by successive transformations of pairs of factors until the sum of the fourth powers no longer increases. See Neuhaus and Wrigley "Brit. J. Statist. Psychol.", Nov. '54.

K14 Vector Element Nonlinear Transformation with Optional Periodic Tape Copy

This program is used to transform experimental data in a variety of ways, preparatory to input to a program such as K15. The transformations of an input x provided are $x^{1/n}, x^{m/n}, c^{-1}(\log_2 x) + 1, c^{-1} \log_2(x + i10^{-p}) + 1, 10^{-a} + 10^{-b}x$.

K15 Analysis of Variance of Symmetrical Orthogonal Experiments

This very versatile program follows the technique for the analysis of variance set out in, for example, Fisher's "The Design of Experiments" (Oliver and Boyd, 1953).

LABEL

TITLE and BRIEF DESCRIPTION

K17

Editing of Data for Analysis with K15

This routine prepares experimental data for analysis by K15, a minimum of initial punching being required.

KA1

Polynomial Approximation

This complete program prints the coefficients of a polynomial which approximates to the function $k f(ax + b)$ with error not greater than e over the range $c \leq x \leq d$.

The function f is defined by the coefficients $f_0, f_1, \dots$ of the Taylor Series.

$$f(y) = f_0 + f_1 y + f_2 y^2 + \dots$$

These coefficients may be either given numerically or generated by an auxiliary subroutine. Any number of coefficients up through $f_{100} y^{100}$ may be given.

A4 (1.7 Precision Floating Binary Arithmetic) is used throughout.

LABEL

TITLE and BRIEF DESCRIPTION

L1 Solution of a Set of Simultaneous Linear Algebraic Equations

This open routine will read a set of simultaneous linear equations punched on the input tape, reduce them to triangular form, and calculate the solutions $x_0, x_1, \dots, x_{p-1}$ by back substitution. If x_n is the scaling factor, the roots will emerge as $x_0/x_n, x_1/x_n, \dots, x_{n-1}/x_n$.

L2 Automatic Linear Equation Solver

This complete program incorporates the Decimal Order Input Routine and Routine L1 above. The programmer need only supply the (scaled) coefficients (of up to 11 decimal digits) of his equations and the number, p , of decimal digits to be printed in the solution. ($p = 1, 2, \dots, 11$.) Scaling during the execution of the program is automatic. Up to 39 equations can be handled, in which case the solution is completed (including input and output) in 4 minutes.

L3 The Complete Linear Equation Solver

This complete program will solve up to 37 simultaneous linear equations using an iteration method. The programmer supplies the scaled coefficients (up to 12 decimal digits) and the number of decimal digits (up to 11) to be printed out in the solution.

L4 Half-precision Solution of a Set of n Simultaneous Linear Algebraic Equations

This open routine will give the solution to, at most, five decimal places of up to 52 simultaneous linear equations, using the triangulation method.

L5 Half-precision Automatic Linear Equation Solver

This complete program incorporates the above routine. In using it the programmer need give only the scaled coefficients of his equations and the number of decimal digits (from 1 to 5 inclusive) he wants in the solutions. Up to 51 equations may be solved. The solution time for 51 equations (including input and output) is less than 8 minutes.

LABEL

TITLE and BRIEF DESCRIPTION

L7 Automatic Solution of More than One Set of Linear Simultaneous Equations

This complete program is a modification of routine L2. Up to 39 equations can be handled in each set, and a number of sets of equations (of varying orders, if required) can be solved seriatim with this program.

L8 Solution of a Set of Linear Simultaneous Equations with a Band Matrix of Coefficients on L.H.S.

This program will solve a set of linear simultaneous equations when the matrix of coefficients is such that all its non-zero elements are contained in a diagonal band. Such equations arise from the finite difference approximation to ordinary or partial linear differential equations. Size restrictions are on p, the number of coefficients in the first row of the non-zero band and on t, the number of right hand sides of the equations. These restrictions are given by $p(p+t) \leq 784$. A set of 100 equations with $p = 11$ may be solved in 6 minutes.

L9 Half-precision Solution of Simultaneous Linear Equations (as in L8)

This program is similar to L8, differing from it in accuracy, the restriction on p and t (given by $(p+1)(p+t+1) \leq 1430$) and in duration (which is approximately half that of L8).

LABEL

TITLE and BRIEF DESCRIPTION

M0	<u>Eigenvalues and Eigenvectors of a Symmetric Matrix</u> This open routine uses Jacobi's method to reduce successively the off-diagonal elements of a symmetric matrix to zero. The eigenvectors of the matrix will be the columns of the diagonalising matrix. Symmetric matrices of order up to 23 can be manipulated by this routine, and by use of special measures, higher order matrices can be handled. Print-out instructions are included. The total time taken depends on the number of iterations required.
M2	<u>Automatic Inversion of a Symmetric Matrix</u> This complete program diagonalises a symmetric matrix A by a series of orthogonal transformations of the type described in M0. If D is the diagonalised matrix and $D = B^T A B$, then the inverse of A is given by $A^{-1} = B D^{-1} B^T$. D^{-1} is readily computed by replacing its diagonal elements by their reciprocals. This program inverted a 23x23 matrix in 14 minutes.
M3	<u>Automatic Eigenvalue - Eigenvector Program</u> This complete program computes the eigenvalues and eigenvectors of a symmetric matrix of order < 24, using M0. For a 23x23 matrix, this program should be completed in less than 13 minutes.
M4	<u>Closed Eigenvalues and Eigenvectors</u> This is a closed routine, a modification of M3, which diagonalises a symmetric matrix (the diagonal elements becoming the eigenvalues). The values of the eigenvectors are computed if required.
M5	<u>Solution of the Determinantal Equation, $A - \lambda B = 0$</u> This complete program solves the equation $A - \lambda B = 0$, where A and B are symmetric and B is positive definite. U is first found so that $B = U U^T$, giving $U^T A U - \lambda I = 0$. The routine diagonalises $U^T A U$, which is symmetric, and the eigenvalues and eigenvectors of the original system are readily computed. The order of the matrices must be less than 18, and for matrices of order 17 the whole process will take less than 6 minutes.

LABEL

TITLE and BRIEF DESCRIPTION

M6	<u>Matrix Multiplication</u> This complete program will multiply together two matrices A and B, where A and B are $(i \times k)$ and $(k \times j)$ matrices respectively, provided $(k \times j) + k \leq 936$. The product matrix $C = (A.B)$ is punched on tape as each element is calculated. The coder may then form D.B by reading in D, or D.E by reading in E, then D. Checks are included which test that $-1 \leq C_{ij} \leq 1$, and that all the rows of A (and B) contain the same number of elements.
M7	<u>Principal Axes Factor Analysis</u> This complete program is a modification of M3. It computes the eigenvalues (i.e. latent roots) and eigenvectors of a symmetric matrix of order up to 23 and punches out the eigenvectors scaled down by the square root of the corresponding eigenvalues. The eigenvectors become factors by this scaling. Duration is the same as M3.
M8	<u>Sums of Squares of Rows and Columns</u> This complete program computes and outputs the sums of squares of the elements in each column of a matrix. The size of the matrix must be such that the number of columns plus twice the number of rows must be less than 920.
M9	<u>Square Root Factorization</u> This complete program uses the square root method to find a factorization of a given correlation matrix. The number of variables must be less than 250. The time taken is about 5 minutes per factor computed.
M10	<u>Estimation of Communalities</u> This complete program will read the matrix of correlation coefficients and compute the communalities.
M11	<u>Matrix Multiplication (Closed)</u> Unlike code M6, this is a closed routine for use in the main program. If A.B is required, there are two possible entries, the first being used when neither A nor B is contained in the storage, the second when B is already stored. Once B has been stored the

LABEL

TITLE and BRIEF DESCRIPTION

M11 (cont.)

procedure is the same as with M6, the same checks as before being available. If B has been stored and A and B are 20x20 matrices, the whole operation is completed in less than half a minute. The product is retained in the store.

M12

Triangulation of a Matrix and Computation of its Determinant

This complete program will take a square matrix of order up to 40, scaled so that each element is less than $1/2$, and triangulate it, using part of code L1 (Solution of Simultaneous Linear Equations). It then computes the determinant. The upper triangle is available in the store and can be punched out if required. The complete computation of a 40x40 matrix will take about 3 minutes of which more than half is input time.

M13

Complete Linear Matrix Equation Solver and General Matrix Inverter

This complete program solves and punches out the solution of the linear matrix equation $AX=B$, where A is non-singular of size (nxn), B is of size (nxm), and both are scaled so that each element is less than $1/2$. n and m must satisfy the inequality: $nm + (n/2)(n+1) + (n+m) \leq 842$. A set of 22 equations will be solved in a total time of 42 seconds, of which about half is computing time.

M14

Linear Matrix Equation Solver and General Matrix Inverter

This closed routine will solve the matrix equation $AC = B$, where A is non-singular of size (nxn), B is of size (nxm) and both are scaled so that each element is less than $1/2$. If B is the unit matrix the routine computes the inverse of A. Auxiliary routines must be provided by the user either to input the matrices A and B or to generate them as required.

M15

Linear Programming

This complete floating point program can be used for the maximization or minimization of linear functions of the form $Z = \sum_{i=1}^n C_i \lambda_i$, subject to m

linear constraints on the λ_i . Constraints which are inequalities must be written as equalities by the introduction of λ_{n+r} , a new λ in each case. An example with 18 constraints which, when converted into equations gave 25 unknowns, was solved in 15 minutes.

M16

Matrix Multiplication when the Common Dimension is Large

This complete program forms the matrix product $A.B$ where A is a $(j \times p)$ matrix and B is a $(p \times k)$ matrix. There is no restriction on the value of p , but j and k are limited in that $j+k \leq 40$. A and B must be scaled so that all elements of A and B are less than unity, and the sums of the squares of the elements in the rows of A and the columns of B must each be less than 10.

M17

Post Multiplication of a Matrix A , or its Transpose A^T , by a Vector k

This closed routine will postmultiply any matrix A (with less than 512 columns) or its transpose (with similar limitations) by a column vector k . The elements A_{ij} of A and k_j of k must be scaled so that

$$\sum_{j=1}^n |A_{ij} k_j| < 1 \quad \text{in the case of } Ak$$

$$\sum_{i=1}^n |A_{ij} k_i| < 1 \quad \text{in the case of } A^T k.$$

For example, this routine would multiply a 1000×500 matrix by a 500 element vector in about 13 minutes.

M18

Automatic Inversion of a Symmetric Matrix with Optional Print Out of Latent Roots and Vectors

This is a complete program and a modification of routine M2. It is limited to matrices no larger than 23×23 . The inverted matrix is punched out in a form which is suitable for later use as the input for other matrix routines. The latent roots and vectors can also be punched out if called for.

M19 Triple Matrix Product $B^T.A.B$ where A is Symmetric
 The triple product will be produced for a series of $(n \times n)$ matrices A_i and an $(n \times j)$ matrix B. Restrictions on j, n, and s are that $nj < 290$, $n \leq 19$, $j \leq 19$, $s < 1024$. A_i are punched as lower triangular matrices row by row, and B^T is punched row by row. All matrices must be scaled such that no element exceeds $n^{-2/3}$ in absolute value.

M24 Matrix Editing Program
 This is a complete program. It outputs the matrix $C = k_1 A + k_2 B$ (or any submatrix of C), where the input of matrices A and B and the output of matrix C may be in various forms. Matrices may thus be converted to forms suitable for input to other matrix routines. If A and B are $(n \times n)$ matrices the routine will process them provided that $2nm + n + m < 704$. If $k_2 = 0$ this inequality becomes $nm + n + m \leq 704$

M25 Eigenvalues of a General Matrix
 This floating point program will output a data tape for use with J2, making possible the computation of eigenvalues of a matrix which may be unsymmetric with real or complex elements. Matrices up to 11×11 can be handled in about 5 minutes of computing time.

MA1 Matrix Multiplication Floating Decimal Auxiliary
 This floating decimal auxiliary multiplies together matrices A $(m \times n)$ and B $(n \times p)$ to give $C = AB$, where m, n and p must satisfy the condition $mp + n + np \leq 825$. To ensure that no element of C exceeds 10^{64} , each element of A and B should be less than or equal to $(\sqrt{10}/n) \times 10^{32}$.
 To multiply two 20×20 matrices already input and store the result, 6 seconds of computing time will be required.

LABEL

TITLE and BRIEF DESCRIPTION

N1	<u>Input One Number from Tape, Integer or Fraction</u> This closed routine will read a sign and the following n digits (e.g. -3333) from the tape and place the signed integer (-3333×2^{-39}) in A and the signed fraction (-.3333) in Q. $n = 1, 2, \dots, 11$. If there is no sign, the number is assumed to be positive. One program parameter is needed.
N2	<u>Input a Sequence of Decimal Fractions</u> This routine reads a sequence of decimal fractions from the tape, where each fraction is punched as a sign (K(+)) or S(-)) followed by up to 12 decimal digits.
N4	<u>Input a Sequence of Integers</u> This closed subroutine reads a sequence of integers from the tape and stores them in consecutive memory locations. Each integer must be punched as a sign followed by up to 12 decimal digits.
N8	<u>Read One Number from Tape as an Integer or Fraction</u> This routine will read a number of n digits from the tape and record it as a signed integer in A, and as a signed fraction in Q. $n = 1, 2, \dots, 12$. This routine is three words longer than N1, but unlike N1, it needs no program parameter, has maximum accuracy and reads 12 digit integers.
N9/P17	<u>Sterling Input and Output</u> A closed subroutine, this will read a sequence of quantities punched as pounds, shillings and pence into consecutive memory locations, or punch out the contents of a specified memory location in the same form. Each quantity must be punched as a sign digit, pounds, any fifth hole character, shillings, fifth hole character, pence, fifth hole character.
N10	<u>Input of a Sequence of Integers with Sum Check</u> This closed routine will input and store in consecutive locations, a sequence of integers punched as a sign followed by up to 12 decimal digits and compute the sum; at a specified point on the tape it will read a punched arithmetic sum and stop if this sum differs from the computed sum.

LABEL

TITLE and BRIEF DESCRIPTION

N11	<u>Input Sequence of Fractions or Integers</u> This closed routine reads a sequence of decimal fractions or integers from the tape, where each number is punched as a sign digit followed by up to 12 decimal digits.
N13	<u>Input of a Sequence of Integers or Fractions</u> This closed subroutine will input and store a sequence of fractions or integers punched as a sign followed by up to 11 digits and provides facilities for a tape stop symbol, an optional check sum, omission of labelled numbers, and simple punching of runs of zeros.
N16	<u>Input of One Number as a Combination of Integer and Fraction</u> This closed routine will read a number consisting of an integral and a fractional part, separated by a decimal point. The integral part must be less than 2^{39} in magnitude, and the fractional part may have have no more than 12 digits.
N17	<u>Input and Output of a Number as a Combination of Integer and Fraction</u> This routine is the same as N16 above, except that in addition, each character read from the input tape is copied onto the output tape.
N18	<u>Input of One Number Containing Decimal Point, Mixed with Headings, with Optional Printing of the Number and the Headings</u> The restriction on the magnitude of the fractional and integral part is the same as in N16 above.
N22	<u>Infraput</u> This fast input routine inputs a sequence of fractions or integers, punched as a sign followed by up to 12 decimal digits. Sequences are terminated by J,F,L,N and thus labelling of different sequences is possible.

LABEL

TITLE and BRIEF DESCRIPTION

N24

Input of a Sequence of Integers

This routine is a fast closed subroutine to input sequences of integers punched as a sign followed by up to 12 decimal digits. Sequences may be labelled by using one of the four terminating symbols N,J,F,L.

N25

Matrix Modifying Input

This closed program parameter inputs a matrix punched in the normal form for input by N2, stores consecutively only non-zero elements, rounded off to 30 bits, the last ten bits indicating the column and row in which the element occurs.

NA26

Input Routine for Use with A7

This closed routine reads a sequence of decimal numbers punched as an integral and fractional part separated by a decimal point, and transforms them into floating binary point form and stores them in successive store locations for use with A7.

LABEL

TITLE and BRIEF DESCRIPTION

P1	<u>Print One Number as a Fraction or Integer, in a Manner Determined by a Program Parameter</u> This routine will punch the number in the accumulator, with or without the sign, as an integer or as a fraction correctly rounded off, as specified by the programmer. Up to 11 digits can be punched and the position of the decimal point can be adjusted.
P2	<u>Print (A) with or without the Sign, to n Places as Determined by a Program Parameter</u> This routine will print or omit the sign, then it will round off and punch the first n places of the contents of the A register, as determined by the parameter.
P3	<u>Print n digits of an Integer with or without the Sign</u> A closed subroutine which prints the integer $X \times 2^{39}$ to n digits, n and the omission or inclusion of a sign being determined by the program parameter.
P4	<u>Zero Suppression Integer Print</u> This closed routine will print (A) as an integer, with or without sign to n places, where n must satisfy $2 \leq n \leq 11$. Zeros on the left will be suppressed.
P5	<u>Print One Number in a Parameter Set Layout</u> The preset parameter is the pseudo order, $O \ C \ bF$ $D_1 D_2 d_3 F$, placed in location 3 before input of the routine. The number in the accumulator will be printed by this closed subroutine as a sign, followed by D_1 digits, space, D_2 digits, space D_3 digits and will be approximately rounded off. The numbers are printed in C columns, each grouped in blocks of b numbers.
P6	<u>Single Column Print</u> This routine will print (A) as a sign followed by n decimal digits, where n is specified by a parameter during the input of this routine. When the number has been printed, carriage return and line feed will be executed.

LABEL

TITLE and BRIEF DESCRIPTION

P7	<u>Letter Printingg</u>	This subroutine will read characters from the input tape and store them in compact form, 8 characters to a store register, and when called upon to do so will print them out again in their original form. Alternatively it will copy the characters straight from the input tape to the output. It is very useful for printing headings and other alphabetical information.
P8	<u>Tape Lettering</u>	This tape lettering routine provides a means of readily identifying output tapes by punching any desired characters in a pictorial form at the start of the tape.
P9	<u>Decimal Order Print Routine</u>	This is a closed subroutine which will print out the order pair from any specified store location.
P10	<u>Print Headings Interlude</u>	This routine is used to print headings above columns of numbers etc.
P11	<u>Rounded or Unrounded Print</u>	This closed routine will print (A) either rounded to n decimals with sign, or to n decimals without round off and without sign. This routine is useful when results are required as an integer part, plus a decimal part without sign.
P12	<u>Single Column Print</u>	This differs from code P6 only in that a delay has been provided after the carriage return.

P13 Combined Integer Print

The manner in which the $(A)x2^{39}$ is punched is specified by the order SY nF, the first of the set of orders which call in this subroutine.

If $X = 5$, a sign is punched;

$X = J$, a sign is not punched;

$Y = 0$, zeros on the left are replaced by spaces;

$Y = 2$, zeros on the left are punched.

The number will be punched mod 10^n , n being the number of significant decimal digits punched.

P14 Print (A) as Integer or Fraction

This closed routine is identical to P1 except in that it prints an actual decimal point whereas P1 prints a space in lieu of a decimal point.

P15 Multiple Precision Integer Conversion

This closed routine will convert a multiple precision integer which has been stored in the form

$$a_0 + a_1(2^{39}) + a_2(2^{39})^2 + \dots$$

with $a_n, a_{n-1}, \dots, a_0$ stored in successive memory locations, to the form

$$b_0 + b_1(x) + b_2(x)^2 + \dots$$

It is normally used with $x = 10^{11}$, the new form of the integer being then ready for printing with a normal integer print routine. The duration is $1.29n^2 + 1.21n$ milliseconds.

P16 Universal Integer/Fraction Print Routine

This closed routine will print the contents of the accumulator as an integer or fraction to any number of digits up to 12, with or without a sign, with or without a decimal point after a specified number of digits, and with or without zero suppression before the decimal point.

P18 Print Headings Routine

This closed routine copies characters from the input tape to the output tape. It is used to print headings, problem numbers, run numbers, etc.

LABEL

TITLE and BRIEF DESCRIPTION

P19	<u>Single Column Fraction Print with Check Sum</u> This closed subroutine will print the contents of the accumulator as a sign followed by up to 11 digits, preceded by a carriage return/line feed symbol. If entered in a special manner, P19 will print the integer sum of all the numbers punched since the previous entry in this manner, i.e. the output of P19 can be used as the input tape for N13 or N15, which will test the check sum.
P21	<u>Combined Integer-Fraction Print with Check Sum and Sequence Numbers</u> This versatile print routine will print one number as a signed fraction of up to 11 decimal places, or a signed integer less than 2^{39} with optional sum check facility as in P19, and facility for punching sequence numbers or matrix row numbers, and sexadecimal terminating symbols.
P24	<u>Double Precision Fraction Print</u> This closed subroutine prints a double length number stored in the form of $x = a + 2^{-39}b$, as a single signed fraction of up to 24 decimals, where the most significant half of the number is stored in location 0, and the least significant half in the Q register.
P25	<u>Optional Printing of Headings</u> This closed subroutine is similar to P18, but can be used when the need to print a given heading depends upon the result of a calculation performed earlier in the program.
P26	<u>Floating Decimal Print Routine for Floating Binary Number</u> This routine accepts numbers of the form $x = a \times 2^p$ where a and p are stored in consecutive locations converts the number to the form $x = b \times 10^q$ where $1 \leq b < 10$ and q is an integer, and outputs b and q.

LABEL

TITLE and BRIEF DESCRIPTION

P27	<u>Single Column Printing of Decimal Fractions from Successive Locations</u> This closed subroutine points successive store locations as signed decimals of up to 11 decimal places in a single column.
P28	<u>Printing of Heading with Specified Terminating Symbols</u> This closed subroutine will copy the input tape onto the output tape until a specified pair of terminating symbols is encountered.
P29	<u>Zero Suppression Integer Print Routine</u> This routine is similar to P4, except it will print any integer except -2^{39} correctly up to 12 decimal places, with or without sign, even though fewer decimal digits are specified for printout than actually occur in the number.
P30	<u>Zero Suppression Integer Printing from Successive Locations in Single Column</u> This routine prints signed or unsigned integers from successive locations in a single column, zero s on the left being replaced by punched spaces.

LABEL

TITLE and BRIEF DESCRIPTION

Q1

Logical Algebra Interpretive Routine

This interpretive routine simplifies the handling of expansions in Boolean algebra. It is intended to aid in the design of switching circuits.

Q2

Logical Input Variable Subroutine

This subroutine generates logical input variables of various types and places them in successive memory locations.

R1 Square Root

This closed subroutine will use Newton's method to compute the square root of a number x of 39 or of 78 binary digits, the sign and the 39 most significant digits being in A, and the 39 least significant digits (if any) being in address 0. When x is $1/4$, the time taken will be about 7 milliseconds.

R2 Integral Root $(A)^{1/p}$

This is a closed subroutine for computing the p -th root of a 39 binary digit number in the accumulator. Newton's method is used. The parameter, p , must be a positive integer satisfying $2 \leq p \leq 1023$. When $p=3$ and $A=0.3$ this will take about 40 milliseconds.

R3 Fractional Power Routine

This closed routine uses the binomial expansion of $(1 + [y-1])^x$ to determine y^x , where x is less than unity in absolute value and must be preset. y is (A) at the beginning of the routine. If y is small, the duration of the calculation is long because of the slow convergence. When y is $2/3$, the time is about 70 milliseconds.

R4 Cube Root

This closed subroutine computes the cube root of the number in the accumulator. Newton's iterative method is used. The time varies from less than 3 milliseconds for (A) greater than $1/4$ to 160 milliseconds if (A) equals 2^{-39} .

R5 Reciprocal

When entered, this closed routine finds the reciprocal of (A), putting the integer part in A and the fractional part in Q. If (A) is of the order of 2^{-n} the time taken is $3 + n/2$ milliseconds.

R6 Fast Square Root

This closed subroutine uses Newton's method to compute the square root of a number of 39 binary digits when the time taken by the routine is of more importance than the store space occupied by it. The time saved by using R6 instead of R1 is appreciable only if the number is less than $1/10$.

LABEL

TITLE and BRIEF DESCRIPTION

R7	<u>Square Root</u> This closed subroutine is similar to R1, except that the argument must be only 39 binary digits. Also, no orders in R7 are modified during operation, and hence R7 may be included in a sum check of the store.
RA1	<u>Floating Decimal Square Root Auxiliary</u> This floating decimal auxiliary will take a number x in the form $x = X \times 10^p$ from the floating accumulator and determine $(1/2)\sqrt{X}$ if p is even and $(1/2)\sqrt{X/10}$ if p is odd. If x is 1/4, the time taken will be about 7 milliseconds.
FA2	<u>Floating Decimal Cube Root Auxiliary</u> This closed auxiliary for the floating decimal routine will replace the number x ($=X \times 10^p$) in the floating accumulator by $\sqrt[3]{x}$. The time taken is about 60 milliseconds.
RA4	<u>1.7 Precision Floating Point Square Root</u> This closed routine is to be used as an auxiliary with A4.

LABEL

TITLE and BRIEF DESCRIPTION

S1	<u>Natural Logarithm</u> This subroutine calculates the natural logarithm of y, $0 < y < 1$, by successively multiplying y by 2 until $1/2 \leq 2^m y < 1$, thereby giving the characteristic of $\log_2 y$, meanwhile forming the number $(-\ln(2^m))/32$. $\ln 2^m y$ is then computed by a Tschebycheff series. If $y=0$ the machine will stop. The time taken will be between 10 and 16 milliseconds.
S2	<u>Exponential</u> This closed routine replaces the number x in the accumulator by e^x if x is negative. The routine takes 12 milliseconds to complete, and uses a short table together with a series.
S3	<u>Logarithm</u> This routine replaces the argument x in A ($0 < x < 1$) by $\log_2 x$ (which is computed by a process of binary chopping). If logarithms to base a are required, the result is converted by the formula $\log_a x = \log_2 x \log_a 2$. The process takes 51 milliseconds.
S4	<u>Exponential</u> This closed subroutine replaces x, the contents of A before entry by e^x. x must lie in the range $-1 < x < 0$. The method used is the evaluation of a continued fraction.
S5	<u>1/32 Natural Logarithm</u> This closed routine replaces x, the contents of A before entry by $1/32 \ln x$ provided $x < 0$. If $x \leq 0$ the machine will stop.
S10	<u>Fast Low Accuracy Logarithm Routine</u> This routine is similar in use and description to S5, though faster. Its duration and accuracy are about one half those of S5.
S11	<u>Fast Exponential Routine</u> This routine is similar to S4, having the same accuracy but half the duration of S4, and is 9 words longer.

LABEL	TITLE and BRIEF DESCRIPTION
S12	<u>Calculation of $2/\sqrt{\pi} \int_0^x e^{-t^2} dt$ or e^{-x^2}</u> This closed subroutine will calculate $2/\sqrt{\pi} \int_0^x e^{-t^2} dt$ in 16 milliseconds, or e^{-x^2} in 9 milliseconds, where $0 < x < 4$.
S13	<u>Fast Low Accuracy Exponential Routine</u> This closed subroutine will compute in 2 to 5 milliseconds 2^x if x is in the A register on entry, or e^x if $x \log_2 e$ is in the A register on entry, provided $-1 \leq x < 6$.
SA2	<u>Exponential Auxiliary for Floating Decimal</u> This floating decimal auxiliary replaces the number $x(=X \times 10^p)$ in the floating accumulator by e^x.
SA3	<u>Natural Logarithm for Floating Decimal</u> This is a subroutine, for use with the floating decimal routine, which will compute the natural logarithm of the number in the floating decimal accumulator in standard form. (If X is the fractional part of the number, $1/10 \leq X \leq 1$.) The subroutine takes 15 milliseconds to complete.
SA6	<u>Floating Binary Point Natural Logarithms</u> This is an auxiliary routine for use with SILLIAC code A7, to compute natural logarithms.
SA7	<u>Floating Binary Point Exponentials</u> This is an auxiliary routine for use with SILLIAC code A7, to compute exponentials.

LABEL

TITLE and BRIEF DESCRIPTION

T1 Sine-Cosine Subroutine

This subroutine uses the number x in the accumulator as argument and places

$(\sin x)/2$ in storage location 1
 $(\cos x)/2$ in storage location 2
 $(\sin 2x)/2$ in storage location 0
 $(\sin 2x)/2$ in Q
 $(\cos 2x)/2$ in A.

The whole process is complete in 20 milliseconds.

T4 Arctan x Subroutine

This closed subroutine takes the number x in the accumulator, and using an economised power series replaces it by $\arctan x$ in 20 milliseconds.

T5 Sine-Cosine Routine

If $\frac{\theta}{\pi}$ is in the accumulator, then this routine will replace it by $(1/2)\sin\theta$ when a standard entry sequence is used.

$(1/2)\cos\theta$ may be obtained by a special entry sequence if θ/π is initially in the store. $\sin\theta$ may be obtained by another special entry sequence if θ and $10/4\pi$ are contained in the store. An economised power series expansion of the sine function is used for computations. This routine is used when not all of the quantities evaluated by code T1 are required. It takes 9 milliseconds to complete.

T6 Fast Sine-Cosine Routine

If x be the number in A on entry to T6, then $\sin x\pi$ will be left in location 0 and A, and $\cos x\pi$ will be left in Q at the completion of the routine. The duration is 10 milliseconds.

T7 Inverse Tangent

Given x in location 0 and y in location 1, this closed subroutine will compute $1/\pi \arctan(y/x)$ and place the result in A.

LABEL

TITLE and BRIEF DESCRIPTION

TA1	<u>Sine Auxiliary for Floating Decimal</u> This floating decimal auxiliary takes the number x from the floating accumulator and replaces it with sin x, the accuracy being small for large x. cos x may be obtained by first adding $\pi/2$ to x. The process takes 13 milliseconds to complete.
TA2	<u>Arctangent Auxiliary to the Floating Point Routine</u> This is a routine for use in a program which contains Library Routine A1 and A3. It takes the argument, x, in the floating accumulator in standard form and replaces it by arctan x. All answers lie in the range $-\pi/2 \leq \arctan x \leq \pi/2$. The process takes 15 milliseconds to complete.

LABEL

TITLE and BRIEF DESCRIPTION

U3 Spherical Bessel Functions

This closed subroutine, when entered with the scaled argument $2^{-n}x$ in A ($0 < n < 16$) will compute and store the spherical Bessel functions $j_0(x), j_1(x) \dots j_K$ where K is the maximum order of the Bessel function to be found and $1 < K \leq 15$. Maximum duration (for $K=15$) is 55 milliseconds.

U4 Associated Legendre Functions

This closed subroutine will compute, for an argument x in A an entry (where $-.985 \leq x \leq .99$, $2^{-19} \leq x \leq .99$) the Associate Legendre Functions $K_0(x), K_1(x), K_1(x), K_n(x)$, where $0 < n \leq 12$, and store them in successive locations determined by a preset parameter.

U5 Ordinary Bessel Functions

With a scaled argument $x \cdot 2^{-m}$ in A on entry, this closed subroutine will compute ordinary Bessel functions of the first, or first and second kinds, and of zero, or first, or zero and first order. Accuracy is about 4 decimal places and the maximum duration is 100 milliseconds.

LABEL

TITLE and BRIEF DESCRIPTION

U1 Legendre Polynomials, $P_n(x)$

This closed routine will take the number x in A and, using a recurrence relation, compute the Legendre Polynomials $P_r(x)$ for $r = 0, 1, \dots, n$, where n is specified. $(1/2)P_0(x), \dots, (1/2)P_n(x)$ will be stored at specified consecutive addresses. The time taken to complete the process is $0.75 + 3.6(n-1)$ milliseconds.

U2 Tschebycheff Polynomials

This routine computes the Tschebycheff polynomials $T_r = 2x T_{r-1} - T_{r-2}$, and places them in successive storage locations. (A) must be $x/16$ when the routine is entered. The duration of the process is $1.3 n$ milliseconds where n is the degree of the $T_r(x)$ of highest degree required.

V3 Random Number Generator

This closed routine generates a sequence of pseudo-random numbers by the method of squaring a number in the sequence and extracting the middle 38 digits to form the next number, and so on. The first 720,000 numbers of this sequence have been tested for randomness. The time taken is 1.7 milliseconds per number generated.

V4 Fourier Analysis

This routine computes approximately $n/2$ coefficients of the sine or cosine expansion of a periodic function, $f(x)$, given by

$$f(x) = \sum_{n=1}^{\infty} A_n \sin n\pi x/L$$

$$\text{or } f(x) = B_0 + \sum_{n=1}^{\infty} B_n \cos n\pi x/L$$

where the parameter n is specified on entry. The time taken is about $(m^2 + 13m)$ milliseconds. A complete Fourier representation of a function may be obtained using this routine.

V5

Fourier Analysis

This program inputs the function $f(x)$ ($-L \leq x \leq L$) as the $(2m+1)$ values.

$$f_{-m} \dots f_{-1}, f_0, f_1, \dots, f_m$$

for a full range series, or alternatively the function $f(x)$ $0 \leq x \leq L$ as the $(m+1)$ values $(f_0, f_1, \dots, f_m)$ for a half range series, and calculates the cosine coefficients $b_0, b_1/2, b_2/2, \dots, b_n/2$

and the sine coefficients $a_1/2 \dots a_n/2$, where

$m \leq 200$ for full range series

$m \leq 320$ for half range series

and $n \leq m$.

V7

2-Dimensional Fourier Synthesis (X-ray crystal structure analysis)

The program sums

$$(x,y) = \sum_{h=0}^{h_{\max}} \sum_{k=0}^{k_{\max}} F_{hk} \begin{pmatrix} \cos \\ \text{or} \\ \sin \end{pmatrix} 2\pi ky/b \cdot \begin{pmatrix} \cos \\ \text{or} \\ \sin \end{pmatrix} 2\pi hx/a$$

where

(x,y) = projected electron density at (x,y) .

F_{hk} = structure factor for $(\underline{hk})$ reflection

$x/a, y/b$ = coordinates, and the summation occurs at intervals of $1/64$ or $1/128$ of each of the unit cell edges.

1280 F_{hk} -terms can be accommodated and may be stored as four separate tables, parameters being supplied which determine the trigonometrical product to be used for each possible condition of $\underline{h}$ and $\underline{k}$.

V8 Structure Factors for Monoclinic Space-groups of the Class C_{2h} .

This program evaluates

$$F_{hkl} = +4 \sum_{j=1}^{N/4} f_j T_f \cos 2\pi(hx_j + lz_j) \cos 2\pi ky_j$$

or

$$-4 \sum_{j=1}^{N/4} f_j T_f \sin 2\pi(hx_j + lz_j) \sin 2\pi ky_j$$

where f_j = atomic scattering factor for the j -th atom,

T_f = an over-all isotropic temperature factor,
 $= \exp(-B \sin^2 \theta / \lambda^2),$

x_j, y_j, z_j = coordinates of j -th atom.

The choice between the two expressions for the structure-factor for each reflection depends on the even/odd-ness of the indices h, k and l . The code for directing this choice consists of a short list of numerical parameters.

V9 Generation of a Sequence of 40 Bit Random Numbers in Groups of Five Numbers

This closed subroutine will generate five of a sequence of pseudo random numbers in 3.4 milliseconds according to the recursion formula.

$$A_{n+5} = 7A_{n+4} + A_{n+3} - 4A_{n+2} + 3A_{n+1} + \pi(A_n)$$

where for a 40 bit number $a_0 \dots a_{39}, \pi(a_0 \dots a_{39}) =$

$$(a_0, a_4, a_5 \dots a_{39}, a_1, a_2, a_3 \dots)$$

V10 Structure Factors for Orthorhombic Space-groups of the classes $222-D_2$, $mm2-D_{2h}$, $mmm-C_{2v}$, excepting $F_{dd2-D_{2h}}$ and $F_{dd2-C_{2v}}$.

This program evaluates $F_{hkl} = A_{hkl}^2 + B_{hkl}^2$ ^{1/2}

where A_{hkl} and B_{hkl} have the form

$$\pm 2^m \sum_{j=1}^{N/2^m} f_j T_f \begin{pmatrix} \cos \\ \text{or} \\ \sin \end{pmatrix} 2\pi h x_j \cdot \begin{pmatrix} \cos \\ \text{or} \\ \sin \end{pmatrix} 2\pi k y_j \cdot \begin{pmatrix} \cos \\ \text{or} \\ \sin \end{pmatrix} 2\pi l z_j$$

and $m = 2, 3$ or 4 (2^m = no. of equivalent general positions),

f_j = atomic scattering factor for j^{th} atom,

T_f = temperature factor

$= \exp(-B_i \sin^2 \theta / \lambda^2)$ where B_i is the thermal parameter of the i^{th} atom-type,

x_j, y_j, z_j = coordinates of j^{th} atom.

LABEL

TITLE and BRIEF DESCRIPTION

V10 (cont.) For non-centro-symmetric space-groups, the potential output consists of $h\ k\ l$, $|F_{hkl}|$, A_{hkl} , B_{hkl} , $\cos \alpha_{hkl}$, $\sin \alpha_{hkl}$.
For centro-symmetric space-groups, the signed value of A_{hkl} corresponds to F_{hkl} .

V11 Generation of Random Normal Deviates

This closed subroutine places a quantity z' in A where $z' = z/8$ and z is a random normal deviate. The probability distribution of z is

$$p(z)dz = 1/\sqrt{2\pi} e^{-z^2/2} dz.$$

If V12 is incorporated as the positive random number generator the duration of V11 is 14 milliseconds/number.

V12 Generation of a Sequence of Positive Random Numbers One at a Time

This closed subroutine is similar to V9, except that the numbers are made positive and generated one at a time in 1.5 milliseconds.

V13 Generation of a Sequence of 40 Bit Random Numbers One at a Time

This closed subroutine is similar to V9 but generates the numbers one at a time in 1.4 milliseconds.

V14 Polynomial Representation of Atomic X-ray Scattering Curves

The SILLIAC crystallographic structure-factor programs V8 and V10 make use of atomic scattering factors $f(\theta)$ which are evaluated as sixth-degree polynomials in $\sin \theta/2\lambda$. The scattering curve for each atomic species is therefore represented in the store by means of the seven coefficients a_n in the expression--

$$f(\theta) = \sum_{n=0}^6 a_n (\sin \theta/2\lambda)^n$$

Polynomial coefficients derived by the use of codes K3 and L7 are tested and tabulated by V14. The representation is generally accurate to better than 1% and is valid only in the range

$$0 \leq \sin \theta/2\lambda \leq 1/2\lambda_{CuK}.$$

LABEL

TITLE and BRIEF DESCRIPTION

V15

X-ray Intensity Correction Factors

This routine computes corrections embracing the Lorentz, polarization and Tunell factors for X-ray intensities recorded by the equi-inclination Weissenberg method. It is applicable to triclinic, monoclinic, orthorhombic, tetragonal and cubic unit-cells. For a single use of the program any number of specified reciprocal lattice layers about one axis will be covered. Any of the quantities $\sin \theta$ (θ = Bragg angle), $\sin \theta / \lambda$ (λ = wavelength of radiation used in collection of data) and $\sin^2 \theta$ for each of the reciprocal lattice points considered in the computation will be output along with the correction factor if so desired.

V16

Provision of Sets of Random Numbers from 1 to N

This complete program stores a sequence of the numbers wanted in a random order prescribed by V3. N may not exceed 900.

LABEL

TITLE and BRIEF DESCRIPTION

W1 Loop Cycling Control

This is a closed routine which will arrange a cycling around a loop of orders any specified number of times up to 127. Several loops may be under control at the same time. Use of this routine eliminates the need to provide a counter within each loop. Time is 1 milli-second for each cycle of a loop.

W2 Word Sorting

This closed routine sorts consecutive words in the memory into order of increasing magnitude. The criterion according to which the words are sorted may be changed by altering a few orders in the routine. The time taken is $1.7n^2$ milliseconds, where n is the number of words to be sorted.

X1 Decimal Order Input

This is an input routine of 25 words, used with nearly every program, allowing programmers to use decimal notation when coding, and facilitating the use of library routines. It converts to binary form orders with addresses in decimal and relative form. For a complete description of this routine, see Chapter 5 of this handbook.

X2 Sum Checking Decimal Order Input

This input routine performs exactly the same function as does the standard decimal order input routine X1. In addition, it forms the sum of all orders read from the program tape. When called upon to do so it will read a check sum from the tape and compare it with the sum it has formed. It thus provides a check on the operation of the tape reader and will indicate when an often used program tape has become distorted or deformed in such a way as to cause it to be misread. The routine is read in by its own bootstrap and occupies store locations 978 to 1023.

X3 Constant Listing Auxiliary

This is an input routine for use with the Decimal Order Input. If a constant is required by the program, the constant itself is used for the address of the order concerned, and the subroutine stores it in a list, placing its address in the order. If the constant is already listed, it is not stored again, the existing address being inserted in the order.

X5 Arithmetic Programming

This is an automatic programming routine which is particularly biased toward the simple and efficient coding of routines for the numerical evaluation of functions such as

$$F = a - b + cd^3 + (e + f) / (g + h) + \sin(j + k)^2.$$

X7 Sum Check

This program, placed at the end of a long tape, is used to check that the program section of a tape has been correctly read before the program starts. It does so by adding all the words in the memory and comparing the sum with the last word of the routine (the sum of all the words on the tape). The check sum is punched out when this routine is first used (in which case, no check sum will have been supplied by the coder).

X8 Block Assembly

A 4-word assembly program, for use with the Decimal Order Input, this permits a programmer to store sub-routines and other blocks making up a program, in successive memory locations without the use of + directives. Each block is then referred to by an S parameter, which is automatically inserted by the subroutine.

X9 Program Interruption Routine (without Sum Check)

This subroutine enables a program to be interrupted at any point, and specified sections of the store to be punched out in sexadecimal. The sections are nominated by the programmer and specified on tape so that it is possible to restart the program later at the point of interruption; when this is required, the master tape is fed in in the normal way, and the tape previously punched out is then taken in by the subroutine, which uses the information contained in it to restore the status quo at the time of interruption.

X11 Sum Check Subroutine

This is a closed routine which enables a periodic sum check to be performed on all or part of the memory during the course of a long program. The correct value of the check sum may or may not be provided by the user, also the value of the sum may be changed a few times in the course of the program. Duration is 1.3 seconds to sum entire store.

X12 Sum Checking D.O.I. with S-parameter, Pre-set Directives and Start Orders

This routine can be used in exactly the same way as X1, but has the extra facilities of a sum check similar to X2, and pre-set directives of the form 00+D and start orders of the form 2BND where D is a sexadecimal character specifying a pre-set S-parameter.

X13 Tag Address Decimal Order Input

This program input routine has normal X1 features plus addition facilities for sequential storing of routines and automatic setting of S-parameters, the ability to tag orders and to specify that any order must be a left- or right-hand order, and the ability to use tag symbols as addresses of orders.

LABEL

TITLE and BRIEF DESCRIPTION

X14 Sum Check Routine

This is a closed subroutine similar to X11, but differs from it in that it does not perform a circular shift of the partial sum on each addition, but simply performs a direct sum of the specified part of the store. It is intended for use where the contents of the summed region are altered by the program.

X15 Decimal Data Input

This routine is used in a manner similar to that of the Decimal Order Input as far as reading in of decimal integers and fractions is concerned, but the preparation of the data is simpler and the read-in is quicker.

X16 Sexadecimal Program Dump with Sum Check

This complete program is used after a program has been read in, but before it has been started, to output the program in sexadecimal form, together with a check sum and start orders, for subsequent rapid, normal bootstrap sexadecimal input.

XA1 Constant Listing Auxiliary for Floating Decimal

This input routine requires for its use both the Floating Decimal Routine and the D.O.I. (which it modifies). When a constant is needed, it is itself used (in the floating decimal form) for the address of the order, and this routine stores it in a list and inserts its address in the order. If it is already listed, it is not stored again, the existing address being inserted in the order.

XA2 Sexadecimal Dump

This routine will convert a decimal program tape to sexadecimal form for faster input. The sexadecimal form of the program takes only 60 per cent of the read in time as the original decimal form. Routine XA2 occupies the same storage locations as the decimal order input X1, viz 999 to 1023. The duration is 5 seconds to read in XA2 and 0.2 seconds to punch out each order pair in sexadecimal form.

CHAPTER 14

DEFINITION OF TERMS

A 2-4[★]

The accumulator register or the contents of the accumulator register.

Access Time

The time to get a number from the memory to a register or the time to get a number from a register to the memory.

Accumulator 2-4

The register into which numbers can be put by addition; same as A register.

Action Cycle

The type of memory cycle in which the memory is connected to the arithmetic unit for the transfer of a number.

Addend 2-5

The number in the memory to be added to the accumulator.

Address 3-2

The right-hand ten binary digits of a twenty digit order; usually the location of a number in the memory.

Analog

Indicates a continuous operation (as opposed to digital).

Arithmetic Unit 2-4

The part of the computer which is required to carry out arithmetic operations.

AQ 3-18

The double -length register including A and Q but excluding the first digit q_0 of the Q register.

Augend 2-5

The number in the accumulator to which another number is to be added.

[★] Numbers refer to pages.

Binary Switch 7-4

A portion of a program which is used to cause two separate sequences of orders to be followed alternately.

Bit 2-1

One binary unit of information or one binary "digit".

Blocking Order 8-7

A transfer order inserted in a program to stop the "normal" sequence of orders so that some check can be made on the program.

Blunder 8-5

A mistake in programming, or more usually a mistake in programming which is most obvious even without running the program.

Boolean

Pertaining to the symbolic logic of George Boole, the mathematician.

Bootstrap Start 5-13

A routine which, with only the aid of the original order pair 80028 and 40000, makes it possible for the machine to continue inputting and storing under the control of orders brought into the machine from the tape.

Carriage Return 9-3

The paper tape code which will cause the typing carriage of a printer to go to the left-hand margin of the paper.

Check Point 8-7

A point in a routine where some check is to be made.

Clear 2-3

The operation of changing all the digits of a register to a common state, usually 0.

Code

A group of more or less arbitrary symbols used to represent some other group of things.

Coding Error 8-1

An error in the detailed preparation of a program.

Collate 3-46

Digitwise logical product, same as extract.

Comparer 9-6

A device for checking the identity of two tapes.

Complement 2-2

See one's complement and two's complement.

Conditional Transfer 3-7

An instruction which will cause a transfer from the pattern of taking orders sequentially provided that a certain condition is satisfied in the machine.

Control 1-3

That part of the machine which serves to "control" the arithmetic unit, the memory and the input-output.

Control Counter 8-5

Usually the order counter.

Control Transfer 3-7

An instruction which may cause a transfer from the sequential pattern for handling orders; see unconditional control transfer and conditional control transfer.

Counter

Usually a short routine which counts the number of iterations carried out in some program.

-D-

Diagnostic Routine

A routine for finding and diagnosing a machine fault.

Digital 1-2

Pertaining to digits; discrete units.

Digitwise-complement 2-2

Same as one's complement; a binary number is the digitwise complement of another binary number if and only if each digit of the number disagrees with the corresponding digit of the other number.

Directive 5-4

An order, usually on a tape, which specifies the location at which a subroutine is to be stored.

D.O.I. 5-1

The decimal order input routine, usually Library Routine X1; used as a general purpose input routine for almost all programs. The other D.O.I.'s are X2, X12 and X13.

Double Precision

Pertaining to numbers of approximately two 40 binary digit number lengths; hence requiring two registers or locations for a single number.

-E-

Error

Mathematically the difference between the correct result and the computed result; often used in place of blunder.

Even Order 3-2

The left-hand order of an order pair, thus using digits 2^0 through 2^{19} .

Extract Order 3-46

An operation which puts a 1 in the Q register wherever the number in the designated memory location and the number in Q are both 1, leaving all other digits zero; also called a logical multiply order because the resulting number in the Q register is the digitwise binary product of the Q register and the number in the memory; same as collate.

-F-

Fixed Address 5-2

The numerical address remains unaltered when put into the memory; usually in connection with D.O.I. where a fixed address is followed by an F.

Fixed Point 2-1

The binary point is always in the same position of a register; SILLIAC is a fixed point machine.

Floating Point

Pertaining to operations in which numbers are represented by a number multiplied by a power of a base; thus numbers may have different multiplying factors; floating point operations in the SILLIAC must be obtained by programming.

Fraction

A number with an absolute value less than 1.

Function Digits 3-2

The first eight binary digits of an order; the T and V digits.

-H-

Hang up

An unplanned stop of the machine due either to a machine fault or a coding error.

-I-

Indexing Order 3-10

An instruction which causes the address of the next following order to be modified by an index number in the control section of the machine, while the address in store remains unaltered.

Input Routine

A routine for inputting other routines, usually aiding in conversion from binary-decimal to binary.

Instruction 3-2

The operation in the machine designated by the first eight binary digits of an order.

Integer

Pertaining to integers; although SILLIAC is a fixed point machine with the binary point between the first two digits it is possible to carry out operations using integer numbers less than $2^{40} - 1$.

Interlude 5-11

A routine which carries out operations and is then destroyed by overwriting as the problem continues, usually carried out during input of information.

Instruction Code 3-1

The order code.

Interpretive Routine 4-12

A routine in which a sequence of operations (instead of a single operation) may be carried out on a number, the sequence designated by parameters carried with the number.

Iteration

A sequence of orders, usually to be executed more than once, and arranged to converge to some analytic result.

-L-

Leapfrog 2-16

An engineering routine which may be used to test the machine.

Left-hand Order 3-2

The left-hand or even order₉ of a pair of orders, thus using digits 2^0 through 2^{19} .

Line Feed 9-3

The paper tape code which causes a Teletype to advance the paper one line; always in conjunction with carriage return.

Link 4-4

The part of the routine used to bring a subroutine into operation and designate the point of return.

Location 1-3

The designation of a number location, address or position in the memory.

Logical Product

When applied to numbers of more than one binary digit, the digitwise logical product; the same as collation of two numbers and the same as extract.

Loop 8-7

A sequence of orders which may be carried out more than once automatically.

-M-

Machine Error

An error caused by the fault in SILLIAC.

Memory 1-2

A device which stores numbers; usually the cathode ray tube or Williams memory in SILLIAC.

-N-

Null Order 3-12

An order which simply sends the SILLIAC on to the next order without any other action.

Number Register 2-5

The temporary location in which the addend, subtrahend, multiplicand and divisor are automatically placed by the control while the corresponding order is being executed; R_3 .

-O-

Odd Order 3-2

The right-hand order of an order pair consisting of the digits 2^{20} through 2^{39} .

One's Complement 2-2

Same as digitwise complement; each digit of a binary number is changed to get its binary complement.

Order

A set of 20 binary or 5 sexadecimal digits which is used to define a machine operation.

Order Code 3-1

A set of sexadecimal characters used to describe the operations in the SILLIAC.

Order Counter 8-5

The counter which keeps a record of the location of the next order pair if no transfer of control is required.

Order Pair 3-3

Two orders which are stored together in one location of the memory.

Order Register 3-6

The register R_3 into which an order pair is transferred from the memory just before either of the orders can be obeyed; R_3 .

Output

A display by paper tape or Teletype printer of the contents of some part of SILLIAC.

Overflow Digits 2-3

The digits which, as a result of a computation, would require digits to the left of 2^0 in a register.

-P-

Page Printer 9-7

An automatic typewriter.

Paper Tape

The common 11/16 inch width paper tape used for handling numbers in and out of SILLIAC.

Parameter 4-9

A number which may be altered from one computation to another but is held fixed during a single run.

Partial Substitution Order 3-27

An order which allows the address digits only to be stored in the memory; a 42 or 46 order.

Plant 4-4

The act of putting some number into an order or routine; usually with reference to an address.

Position

Referring to one of the 2^0 to 2^{-39} places of the memory or of a machine register.

Post Mortem 8-9

The checking of the routine in the memory after it has been used, usually by an automatic routine which compares the contents of the memory with the original input tape.

Program 1-4

The plan of a calculation.

Punch 9-1

A unit to punch holes in 11/16 inch paper tape, usually **punch** providing output from the SILLIAC.

Q or Q Register 2-4

The register which must be used for the multiplier during a multiplication; also the register into which the quotient is placed during division; the contents of the Q register.

Quotient Register 2-4

Same as Q register.

Quadrature

The process of evaluating a definite integral by numerical means.

R₃ 3-4

The order register; bottom row of 40 neon lights on SILLIAC.

Range

The set of numbers which may be handled in a SILLIAC register directly from and including -1, up to and including $1 - 2^{-39}$.

Raster

The array on a memory tube.

Read-around

An index of the interference of one spot with another in the Williams memory.

Reader 9-1

The device to take data from a punched paper tape for insertion into SILLIAC.

Regeneration

The process of refreshing the stored information on a Williams tube.

Register 2-5

A row of 40 flip-flops which can hold a binary number.

Relative Address 5-2

An address which is relative to a directive and hence must be added to the directive address to get the true memory location.

Remainder 2-12

The quantity left from the dividend after a division process (without regard to SILLIAC).

Residue 2-13

The quantity in A after a division order has been obeyed (in the SILLIAC).

Right-hand Order 3-2

The order using positions 2^{-20} through 2^{-39} ; same as odd order.

Round-off 2-12, 13

The process of adding 2^{-40} in multiplication and making $q_{39} = 1$ in division when the term is used with SILLIAC.

Routine 4-1

A completed sequence of orders in coded form.

-S-

Scale 6-1

The adjustment of a number to come within the range of SILLIAC.

Sexadecimal 3-3

A number system with base 16.

Shift

The process of moving a number to the right or left in a register.

Sign Digit 2-1

The first position in a register; 2^0 .

Single Address Code

A machine order code which has one address only with each order; SILLIAC has a single address code.

Store

The memory, usually Williams memory.

Subroutine 4-1

A routine arranged according to a standard pattern so that it can be easily used as part of other routines.

Sum Check 5-22

The sum of all the orders and pseudo-orders read from a program tape, which can be compared with a sum provided at the end of the tape, to ensure that the program has been read in correctly by SILLIAC.

Slave Tube 8-7

A device for displaying the entire contents of one memory cathode ray tube.

-T-

Tape Code 9-1

The hole patterns in a paper tape that are used to represent numbers and instructions.

Tape Comparer 9-6

A device to compare two paper tapes.

T Digit 3-13

The first sexadecimal digit of an instruction.

Temporary Storage

Locations in the memory used briefly during a calculation and not assigned to any number to be retained for results.

Terminating Symbol

A symbol on the tape indicating the end of a section of code or tape.

Two's Complement 2-2

The difference between 2 and the number whose two's complement is to be found; in the SILLIAC identical, because it is modulo 2, to reversing each digit and adding 2^{-39} .

-U-

Unconditional Transfer 3-7

A transfer out of the ordinary sequential pattern of handling orders.

-V-

V Digit 3-13

The second sexadecimal digit of an instruction.

Waste Order 4-3

An order that serves no computational purpose but is inserted because of ease in coding or limitation of the machine because of odd and even pairing of orders.

Williams Tube

A cathode ray tube used for storing binary digits.

Word

Forty binary digits in a register or a single location in the memory.

Working Space

See temporary storage.

Appendix I

BINARY AND SEXADECIMAL NUMBERS

AND THEIR MANIPULATION

I.1. POSITIONAL NOTATION. The normal way of writing a decimal integer $x_n x_{n-1} \dots x_0$, is a shorthand form of

$$\sum_{r=0}^n x_r 10^r.$$

Thus, 231 is $2 \times 10^2 + 3 \times 10^1 + 1 \times 10^0$. The base, or radix, as it is sometimes called, need not be 10, a choice which has resulted from the number of fingers on our hands. Moreover, it need not be fixed from one digit to the next. Thus in £.s.d. the bases are 10,2,10,12 and £21.13.2 is thus

$2 \times 10 \times 2 \times 10 \times 12 + 1 \times 2 \times 10 \times 12 + 1 \times 10 \times 12 + 3 \times 12 + 2$
pence. Such a representation is known as a mixed radix representation.

Fractions in the positional notation are represented similarly. Thus $x_1 x_2 \dots x_n$ in base b would represent the fraction

$$\sum_{r=1}^n x_r b^{-r}.$$

I.2. MOST ECONOMICAL BASE. To express N numbers in base b requires n digits, where

$$n \doteq \log_b N = \ln N / \ln b,$$

$\doteq$ having the meaning: " n is the next largest integer to", if $\log_b N$ is not an integer, "equals" if $\log_b N$ is integral. We assume that the number of engineering elements required to represent a single digit in base b is proportional to b . Thus the number of components required to represent n digits in base b is C where

$$C = k n b,$$

where k is a constant. We wish to minimise $k n b$, the number or components required to represent a fixed number of numbers N (which we take as b^n).

Thus

$$C = k b \ln N / \ln b,$$

$$= K b / \ln b,$$

where $K = k \ln N$.

$$dC/db = K(\ln b - 1)/(\ln b)^2.$$

This^(c) is a minimum where $\ln b = 1$, i.e. $b = e$.

I.3. BINARY, OCTAL, SCALE-of-16 and SCALE-of-32 REPRESENTATIONS

In practice, the natural engineering advantages of $b = 2$ (the two digits being represented by a valve conducting or non-conducting, a switch open or closed, remanent magnetism in one direction or another, etc.) gives the binary base an overwhelming advantage, so that, even when other considerations call for different bases--e.g. a decimal base, which would avoid inter-radix conversion in many cases--it is usual to construct the circuitry from combinations of binary switches. Exceptions to this rule occur in the case of several multi-electrode valves which are available on the market, which enable one to count direct in scale of 10. In binary, the integer $x_n \dots x_0$, where x_r is either 0 or 1, represents the number

$$\sum_{r=0}^n x_r 2^r.$$

The first ten integers are thus as follows:

<u>Decimal Representation</u>	<u>Binary Representation</u>
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010

To represent N decimal digits would require B binary digits, where

$$B \doteq N \log_2 10 \approx 3.4N$$

Clearly for the recording and verbal repetition of numbers in binary, the use of binary digits ("bits") alone is unwieldy. This defect is overcome by grouping the bits of a binary number into groups of three, four or five, and allocating 8, 16 or 32 symbols to the resulting 2^3 , 2^4 or 2^5 combinations.

The first arrangement, known as the octal system, has, it is claimed, the advantage that no symbol in addition to the first eight decimal digits (0 - 7) is required for the representation of 2^3 combinations. It is widely used, and some desk calculators have been constructed to work in this radix, as an adjunct to a binary digital computer.

The second arrangement, known as the sexadecimal representation, is used with SILLIAC (where, in addition to the symbols 0 - 9, the symbols + (or K), - (or S), N, J, F, L are used to represent the equivalents of 10, 11, 12, 13, 14, 15). The sexadecimal system is claimed to have the advantage over the octal system that, as symbols other than the decimal digits are present, there is little chance of mistaking a number written in sexadecimal for a decimal number. (Note that "sexadecimal" really means "scale of 60", although usage has sanctioned the meaning given here.) The scale-of-32 representation is used by the Manchester University group, where all of the 32 symbols are assigned independently of any other system.

I.4. INTER-RADIX CONVERSION - INTEGERS. It is possible to convert an integer expressed in base b to an integer expressed in base d by successive divisions by d. The remainders give successive digits, starting from the least significant. Thus, to express 21 in binary, we have:

$$\begin{aligned}
21/2 &= 10 + \underline{1}/2 \\
10/2 &= 5 + \underline{0}/2 \\
5/2 &= 2 + \underline{1}/2 \\
2/2 &= 1 + \underline{0}/2 \\
1/2 &= 0 + \underline{1}/2,
\end{aligned}$$

leading to 10101 (the underlined digits) as the representation of 21 in binary.

It is often more convenient to obtain the most significant digit first. If we know that the number we are converting is always less than d^{n+1} , then successive divisions by d^n , d^{n-1} , ..., will give us the converted number, successive quotients giving the converted digits. To avoid storing (or computing) d^n , d^{n-1} , etc., the same effect can be achieved if we continue to divide by d^n , successive remainders being multiplied by d .

Thus, if we wish to convert 3LL (sexadecimal) to decimal, and we know that the converted number is less than 10,000 (decimal), we divide by 3F8, the equivalent of 1,000 (decimal), as shown below. (Binary division and multiplication follows the same procedure as division and multiplication in decimal. These processes will be fully discussed when arithmetic processes are detailed in a later section.)

$$\begin{aligned}
3LL/3F8 &= \underline{1} + 017 \\
017 \times &= 0F6 \\
0F6/3F8 &= \underline{0} + 0F6 \\
0F6 \times &= 8LN \\
8LN/3F8 &= \underline{2} + 12N \\
12N \times &= --8 \\
--8/3F8 &= \underline{3} + 000
\end{aligned}$$

As a check, the final remainder must always be 000. This process is important when a number is being converted from binary to decimal for output from a binary computer.

When information is being input to a computer digit by

digit, we should have a process of converting information which is presented digit by digit, most significant digit first. Use is made of the full expansion of the number to be converted.

$$\text{Thus } x_n x_{n-1} \dots x_0 = \sum_{r=0}^n x_r d^r.$$

This series may be generated with the help of the algorithm:

$$\begin{aligned} u_{r+1} &= d u_r + x_{n-r}, \\ u_0 &= 0. \end{aligned}$$

Thus, in converting 1023 (decimal) to binary, we have, in sexadecimal,

$$u_1 = 001, u_2 = 00+, u_3 = 066, u_4 = 3LL.$$

I. 5. INTER-RADIX CONVERSION - FRACTIONS. A fraction expressed in base b can be converted to base d by successive multiplications by d and extraction of the integers in the resulting products.

Thus, if we have a number which is 0.1101 in binary (i.e. 13/16), successive multiplications by 10 (1010 in binary) yield the products shown below. Successive decimal digits (in binary) are underlined.

0.1101	Original number
<u>1000</u> .0010	Product by ten
<u>1</u> .0100	Product of fractional part by ten
<u>10</u> .1000	"
<u>101</u> .0000	"

The decimal equivalent is thus 0.8125.

The justification for the algorithm rests on the following observation. If N is the fraction to be converted, in base d it will take the form

$$\sum_{r=1}^n a_r d^{-r}.$$

Thus d N is

$$\sum_{r=0}^{n-1} a_{r+1} d^{-r},$$

the first term of which is a_1 , the first converted digit. The argument is repeated for subsequent digits.

If a fraction is input to a computer in base 10, most significant digit first, it can be converted to binary by multiplying successive digits by $1/10$, $1/100$, $1/1000$ etc., these fractions being expressed in binary. An alternative technique is to present the numbers l.s.d first (for example, by reversing the tape on which they were punched), and recall that $a_1 a_2 \dots a_n$ represents the number

$$(\dots(((a_n)d^{-1} + a_{n-1})d^{-1} + a_{n-2})d^{-1} + \dots + a_1)d^{-1}.$$

Thus conversion takes the form of continued applications of the recurrence relation

$$u_{r+1} = d^{-1} u_r + a_{n-r}$$

where $u_0 = 0$, $a_0 = 0$. The process usually terminates with the detection of the sign symbol preceding the number (i.e. following it on input).

Integer input and output arrangements can be used for fractions if suitable adjusting factors are used. Thus a 10 decimal digit fraction, converted to binary as an integer, would have to be multiplied by 10^{-10} to achieve the correct result. A binary fraction would have to be multiplied by the appropriate power of 10 before an integer conversion scheme could be used. Mutatis mutandis, similar remarks apply when fractional conversion schemes are to be used to convert information representing integers. Because conversion factors cannot be represented exactly in general, the final result may be in error in the last place.

Similar adjustments can be made with a number which has the point separating fractional and integer parts lying somewhere in the centre, so that either a fractional or an integer conversion scheme can be used. Note that all scaling factors must be adjusted so that the capacity of the arithmetic arrangements is not exceeded, and as little

accuracy as possible is lost. Thus the factor 10^{-10} mentioned in the previous paragraph would appear in the SILLIAC as $2^{36}/10^{10}$ (i.e. 0.687 ...), the exponent 36 being chosen so that the factor is as close to 1 as possible, but less than it.

I 6. REPRESENTATION OF NEGATIVE NUMBERS. Consider an arithmetic register which is capable of holding say five bits. It is possible to reserve the first bit position for sign, and to have the remaining four bits represent the absolute value of the number. This is the normal arrangement in writing numbers: it lends itself easily to the construction of multiplication and division circuits, as positive factors or divisors and dividends only have to be catered for, signs being adjusted separately. However, addition and subtraction circuits are more complicated with this arrangement than with the ones to be described, which are commonly adopted in computer circuitry.

The alternative representation is to preserve the most significant bit to represent sign, but in the case of negative numbers to carry out one of two operations on the remaining bits. The first of these is to write each bit as the negation of the corresponding representation of the positive number. The second is to write a negative number $-x$ as $K-x$, where K is the number represented by a 1 in the position beyond that of greatest significance.

Suppose we have a five-bit register in which the binary point is considered to be between the first and the second bit position. Here, 0.1000 (binary) represents 0.5 and the first of the alternative representations is to represent -0.5 as 1.0111. Such a representation is known as the "1's complement representation". If we add together the representations of $+0.5$ and -0.5 , we obtain 1.1111, which

is a representation of zero. An alternative representation is, of course, 0.0000: this dual representation of zero is a peculiarity of 1's complementing. A second feature is that, in adding two binary numbers together, a consistent representation is obtained only when we add a carry beyond the most significant end into the least significant. Thus $3/4 - 1/2$ is $0.1100 + 1.0111$, which, on application of this treatment of carry, yields the correct result, 0.0100. Such an arrangement is known as "end-around-carry". It is a legacy from the construction of punched-card-calculating machines operating in parallel in scale-of-10; this form of representation is seldom used today. By its nature, it is not suitable for serial devices, in which the serial treatment of bits makes "end-around-carry" expensive to achieve.

The other arrangement, known as "2's complement representation" is the one used in SILLIAC.

A simple rule for writing down the 2's complement of a number is the following. Proceeding from the least significant end, copy all zeros in the number until the first 1 is encountered, which is also copied. Thereafter, all digits are copied in their negated form.

Thus the 2's complement of

00110101000

is 11001011000,

the 1's being terminated at the m.s.d., this being the sign indicator. This rule for conversion lends itself readily to the construction of suitable logical circuits. It may be stated alternatively by saying that the 2's complement is obtained by adding 1 in the l.s. place to the 1's complement.

I. 7. SCALING, FIXED AND FLOATING POINT. From this point on, it will be assumed that $-x$ is represented as $K-x$, where K is

the number represented by a 1 in the position beyond that of greatest significance, i.e. Z in SILLIAC.

In scientific calculations, the positioning of the point is apt to cause considerable concern. In simpler computing devices, the position of the point is considered to be fixed, and any necessary scaling is taken care of by a multiplication by powers of 2 in a binary machine (i.e. a shift to the right or left) or 10 in a decimal machine, or by the introduction of other adjusting factors.

In fixed point machines, the position of the point is of no concern in the processes of addition and subtraction. However, the detailed construction of the multiplier requires some decision in this regard, a point to which we shall return when we come to discuss binary multiplication. Common choices for general purpose machines are expressed by giving the range of numbers held. These are:

$$(a) \quad 1 > x \geq -1$$

$$(b) \quad 2 > x \geq -2$$

$$(c) \quad 2^n > x \geq -2^n \text{ (for one } n+1 \text{ - bit register).}$$

The first, or fractional form of representation, is the usual one: it corresponds to the positioning of a binary point between the first and second bits. The slight asymmetry in the range stems from the fact that $0.111 \dots$ is, in an n -bit register, $1-2^{-(n-1)}$, which is the largest positive number which can be held, whereas $1.00 \dots$ is -1 , the largest negative number which can be held. Note that multiplication of fractions results in increasing length in the direction of decreasing significance, i.e., although multiplication cannot lead to overflow, the retention of the first n bits of the product of two n -bit factors can cause a loss of significant bits.

The second possibility has been adopted quite widely,

and has two advantages. The range includes the number +1, a defect of representation (a) which may sometimes be found irritating. Moreover, the programming of operations calling for an adjustment of scaling factors when overflow is anticipated is done most conveniently with numbers normally held in range (a), which can also occupy values such that $2 > |x| \geq 1$ without loss of information. After an addition (or subtraction) operation with two numbers in range (a), a test to see if they are in range $2 > |x| \geq 1$, followed by halving if this is so, affords an easy means of control.

The last possibility (c) may be considered as being equivalent to a representation in which $1/2 > x \geq -1/2$. For consider the process of multiplication when, to retain the product of two n-bit factors, a 2n bit product register is provided. In integer representation, a number 0100 ... represents 2^{n-2} . If we square this, we obtain 2^{2n-4} , which is represented as 000100.... in the 2n-bit product register. This interpretation is consistent with taking the factors as 1/4, the product as 1/16. Note that treating numbers as integers leads to growth in the upward direction, a fact which complicates scaling.

A program can be arranged so that numbers are represented by number pairs, such that the number pair (a,b) represents $a \times 2^b$ or $a \times 10^b$ where $1 > |a| \geq 1/2$ or $1 > |a| \geq 1/10$, $0 < |b| < 2^n$ or 10^n . Also, machines are commonly constructed with this form of representation built into the hardware. The form of representation is known as "floating point" or "semi-logarithmic".

I. 8. ADDITION AND SUBTRACTION. Addition and subtraction in floating point form call for initial scaling when there is a difference of exponents, as well as a subsequent adjustment to ensure the sum or difference is in the standard form of representation.

The treatment of fixed point subtraction is usually carried out, not with a special subtracting unit, but by passing the subtrahend through a complementer and then using an adder.

Terms in use in describing the numbers involved in addition and subtraction operations $A \pm B$ are "augend" or "diminuend" for A, "addend" or "subtrahend" for B.

Overflow detection after addition is conveniently carried out by carrying an extra bit beyond the most significant in augend, addend and sum. If this bit does not agree with the one of next lower significance, overflow has occurred.

Thus, if we add $1/2$ to $1/2$, we have in fractional representation

$$\begin{array}{r} 0 \text{ ' } 0 \ 1 \ 0 \ 0 \ \dots \\ 0 \text{ ' } 0 \ 1 \ 0 \ 0 \ \dots \\ \hline 0 \text{ ' } 1 \ 0 \ 0 \ 0 \ \dots \end{array}$$

which is $+1$, a number outside the range of the register, However, if we add $-1/2$ to $-1/2$, we have

$$\begin{array}{r} 1 \text{ ' } 1 \ 1 \ 0 \ 0 \ \dots \\ 1 \text{ ' } 1 \ 1 \ 0 \ 0 \ \dots \\ \hline 1 \text{ ' } 1 \ 0 \ 0 \ 0 \ \dots, \end{array}$$

which is -1 , a number within the range of the register.

I.9. BINARY MULTIPLICATION AND DIVISION. In what follows, we shall assume that numbers are held in the fractional representation, so that the binary point appears between first and second bits.

In this representation, if a $2n$ -bit register is provided to hold the product of two n -bit factors, the least significant bit is never used. That this is so is clear from the fact that, if we square 2^{-n+1} (i.e. a number with 1 only in the least significant bit of the factors), the result is 2^{-2n+2} , which is a number with a 1 only in the

bit position next to the least significant of the $2n$ -bit register.

This fact is made use of in machines such as SILLIAC to arrange for the multiplier to operate in the fractional mode, and at the same time to multiply small integer factors represented in n -bit registers scaled by 2^{-n+1} so that, if the second half of the $2n$ -bit product is taken, the representation of the product is consistent (e.g. if the factors are x & y , these will be represented in 40-bit registers as $2^{-39}xy$). This is arranged for by designing the multiplier so that the bit which is always zero appears at the beginning of the second half of the $2n$ -bit product register.

Thus, consider the case ^{products of} of numbers held in 5-bit registers. In the normal representation, $2^{-4} \times -2^{-4}$ will appear as

$$0.0001 \times 1.1111 = 1.1111/11110$$

the slant appearing between the two halves of the product.

In the modified arrangement, the product will appear as

$$1.1111/(0)1111$$

the most significant bit of the less significant half appearing in brackets. To illustrate what happens when we take the product of two small integers, consider 1×1 . These will be represented as $2^{-4} \times 2^{-4}$, which will appear as

$$0.0000/(0)0001,$$

i.e. 2^{-8} . The second half of this product is 2^{-4} , the representation of the product.

The treatment of negative factors can be determined by writing down their representations--as is set out in the following table.

Product	Represent- ation of Factors	Product of Representations	Represent- ation of Product	Correction
$x \times y$	x, y	xy	xy	-
$-x \times y$	$K-x, y$	$Ky - xy$	$K^2 - xy$	$K(K-y)$
$x \times -y$	$x, K-y$	$Kx - xy$	$K^2 - xy$	$K(K-x)$
$-x \times -y$	$K-x, K-y$	$K^2 - K(x+y) + xy$	xy	$K(x+y)$

Consider, as examples, the product of 5-bit integer factors. Here $K = 2^5$. Examples of the four products mentioned above, with comments, follow.

$$\begin{array}{r}
 x \times y \qquad \qquad \qquad 3 \times 5 \\
 \qquad \qquad \qquad \qquad \qquad 00011 \\
 \qquad \qquad \qquad \qquad \qquad 00101 \\
 00000 / 00011 \\
 00000 / 011 \\
 \hline
 00000 / 01111 = 15.
 \end{array}$$

$$\begin{array}{r}
 -x \times y \qquad \qquad \qquad -3 \times 5 \\
 \qquad \qquad \qquad \qquad \qquad 11101 \\
 \qquad \qquad \qquad \qquad \qquad 00101 \\
 00000 / 11101 \\
 00011 / 10100 \\
 \hline
 00100 / 10001 \\
 11011 / \qquad \qquad \qquad 2^5(2^5-5) \\
 \hline
 11111 / 10001 = -15.
 \end{array}$$

Alternatively, if the most significant bit of partial products is extended (i.e. repeated), no correction is necessary. For $-x$ is then represented as $K^2 - x$, and the product, $K^2 y - xy$, is the same as $|K^2 - xy| \text{ modulo } K^2$ ¹ --i.e. has the same representation in a $2n$ -bit register.

Detailed operations in our example are then as follows:

$$\begin{array}{r}
 11101 \\
 00101 \\
 \hline
 11111 / 11101 \\
 11111 / 10100 \\
 \hline
 11111 / 10001 = -15
 \end{array}$$

$$\begin{array}{r}
 x \times -y \qquad \qquad \qquad 3 \times -5 \\
 \hline
 \end{array}$$

¹ In computer jargon, $|x| \text{ modulo } y$ is the remainder when x is divided by y --i.e. $x = yk + |x| \text{ modulo } y$. This is not quite the same as the meaning ascribed to $|x| \text{ modulo } y$ in textbooks on number theory.

$$\begin{array}{r}
00011 \\
11011 \\
\hline
00000/00011 \\
00000/00110 \\
00000/11000 \\
00001/10000 \\
\hline
00010/10001 \\
11101/ \\
\hline
11111/10001
\end{array}
\quad
\begin{array}{l}
2^5(2^5-3) \\
=15
\end{array}$$

The last two stages could have been combined if we had subtracted 3×2^4 instead of adding it.

$$-x \times -y \quad -3 \times -5$$

$$\begin{array}{r}
11101 \\
11011 \\
\hline
00000/11101 \\
00001/11010 \\
11000/01000 \\
01110/10000 \\
\hline
11000/01111 \\
00011/00000 \\
00101/00000 \\
\hline
00000/01111
\end{array}
\quad
\begin{array}{l}
2^5 \times 3 \\
2^5 \times 5 \\
=15
\end{array}
\quad
\begin{array}{l}
x \\
x
\end{array}$$

Of course, the devices for shortening $-x \ y$ and $x \ -y$ can both be employed here.

The detailed construction of binary dividers follows along similar lines, and will not be discussed here. The exact form of quotient and residue in the case of SILLIAC are discussed in Chapter 3 (q.v.).

Base 10	Base 16	Base 10	Base 16
0	000	512	200
16	010	528	210
32	020	544	220
48	030	560	230
64	040	576	240
80	050	592	250
96	060	608	260
112	070	624	270
128	080	640	280
144	090	656	290
160	0A0	672	2A0
176	0B0	688	2B0
192	0C0	704	2C0
208	0D0	720	2D0
224	0E0	736	2E0
240	0F0	752	2F0
256	100	768	300
272	110	784	310
288	120	800	320
304	130	816	330
320	140	832	340
336	150	848	350
352	160	864	360
368	170	880	370
384	180	896	380
400	190	912	390
416	1A0	928	3A0
432	1B0	944	3B0
448	1C0	960	3C0
464	1D0	976	3D0
480	1E0	992	3E0
496	1F0	1008	3F0

Table I.1.

Decimal-Sexadecimal Address Conversion

Appendix II

TABLE OF POWERS OF 2

				2^n	n	2^{-n}																				
				1	0	1.0																				
				2	1	0.5																				
				4	2	0.25																				
				8	3	0.125																				
				16	4	0.0625	5																			
				32	5	0.03125	25																			
				64	6	0.015625	625																			
				128	7	0.0078125	8125	5																		
				256	8	0.00390625	90625	25																		
				512	9	0.001953125	953125	125																		
	1	024		10	0.0009765625	9765625	5625	5																		
	2	048		11	0.00048828125	48828125	28125	25																		
	4	096		12	0.000244140625	244140625	140625	625																		
	8	192		13	0.0001220703125	1220703125	0703125	3125	5																	
	16	384		14	0.00006103515625	06103515625	03515625	15625	25																	
	32	768		15	0.000030517578125	030517578125	517578125	578125	125																	
	65	536		16	0.0000152587890625	0152587890625	2587890625	7890625	0625	5																
	131	072		17	0.00000762939453125	00762939453125	62939453125	39453125	53125	25																
	262	144		18	0.000003814697265625	003814697265625	814697265625	697265625	265625	625																
	524	288		19	0.0000019073486328125	0019073486328125	9073486328125	3486328125	6328125	8125	5															
	1	048	576	20	0.00000095367431640625	00095367431640625	95367431640625	67431640625	31640625	40625	25															
	2	097	152	21	0.000000476837158203125	000476837158203125	476837158203125	837158203125	158203125	203125	125															
	4	194	304	22	0.0000002384185791015625	0002384185791015625	2384185791015625	4185791015625	5791015625	1015625	5625	5														
	8	388	608	23	0.00000011920928955078125	00011920928955078125	11920928955078125	20928955078125	28955078125	55078125	78125	25														
	16	777	216	24	0.000000059604644775390625	000059604644775390625	604644775390625	644775390625	775390625	390625	625															
	33	554	432	25	0.0000000298023223876953125	0000298023223876953125	8023223876953125	3223876953125	3876953125	6953125	3125	5														
	67	108	864	26	0.00000001490116119384765625	00001490116119384765625	90116119384765625	16119384765625	19384765625	84765625	65625	25														
	134	217	728	27	0.000000007450580596923828125	000007450580596923828125	450580596923828125	580596923828125	596923828125	923828125	828125	125														
	268	435	456	28	0.0000000037252902984619140625	0000037252902984619140625	7252902984619140625	2902984619140625	2984619140625	4619140625	9140625	0625	5													
	536	870	912	29	0.00000000186264514923095703125	00000186264514923095703125	86264514923095703125	64514923095703125	14923095703125	23095703125	95703125	03125	25													
1	073	741	824	30	0.000000000931322574615478515625	000000931322574615478515625	931322574615478515625	322574615478515625	574615478515625	615478515625	478515625	515625	625													
2	147	483	648	31	0.0000000004656612873077392578125	0000004656612873077392578125	4656612873077392578125	6612873077392578125	2873077392578125	3077392578125	7392578125	2578125	8125	5												
	4	294	967	296	32	0.00000000023283064365386962890625	00000023283064365386962890625	23283064365386962890625	83064365386962890625	64365386962890625	65386962890625	62890625	90625	25												
	8	589	934	592	33	0.000000000116415321826934814453125	000000116415321826934814453125	116415321826934814453125	415321826934814453125	321826934814453125	826934814453125	814453125	453125	125												
17	179	869	184	34	0.0000000000582076609134674072265625	0000000582076609134674072265625	0582076609134674072265625	2076609134674072265625	2076609134674072265625	6609134674072265625	9134674072265625	4674072265625	2265625	5625	5											
34	359	738	368	35	0.00000000002910383045673370361328125	000000002910383045673370361328125	02910383045673370361328125	10383045673370361328125	10383045673370361328125	83045673370361328125	45673370361328125	73370361328125	70361328125	61328125	28125	25										
68	719	476	736	36	0.000000000014551915228366851806640625	0000000014551915228366851806640625	014551915228366851806640625	551915228366851806640625	915228366851806640625	228366851806640625	366851806640625	851806640625	806640625	640625	625											
137	438	953	472	37	0.0000000000072759576141834259033203125	00000000072759576141834259033203125	0072759576141834259033203125	2759576141834259033203125	9576141834259033203125	6141834259033203125	1834259033203125	4259033203125	9033203125	3203125	3125	5										
274	877	906	944	38	0.00000000000363797880709171295166015625	000000000363797880709171295166015625	00363797880709171295166015625	63797880709171295166015625	97880709171295166015625	80709171295166015625	09171295166015625	71295166015625	95166015625	66015625	15625	25										
549	755	813	888	39	0.000000000001818989403545856475830078125	0000000001818989403545856475830078125	001818989403545856475830078125	818989403545856475830078125	989403545856475830078125	403545856475830078125	545856475830078125	856475830078125	475830078125	830078125	078125	125										

Appendix III

USEFUL CONSTANTS

π	3.14	159	265	359
$\pi/2$	1.57	079	632	679
$\sqrt{\pi}$	1.77	245	385	091
π^2	9.86	960	440	109
$1/\pi$	0.31	830	988	618
$360/2\pi$	57.2	957	795	131
e	2.71	828	182	846
$\log_e \pi$	1.14	472	988	585
$\log_e 10$	2.30	258	509	299
$\log_e 2$	0.69	314	718	056
$\sqrt{2}$	1.41	421	3 56	237
$\sqrt{3}$	1.73	205	080	757
$\sqrt{10}$	3.16	227	766	017
$\log_2 10$	3.32	192	809	489
$\log_{10} 2$	0.30	102	999	566

APPENDIX IV

READ AROUND RATIO

What "Read Around Ratio" Is.

Numbers and instructions are stored in the SILLIAC on the screens of cathode ray tubes. A cathode ray tube is, of itself, capable of storing information, in the form of a pattern of positively charged spots on its screen, only for a very short time; about $1/10$ second at most. To obtain long time storage in a computer it is necessary for the machine to contain circuits which cause the information to be read and regenerated regularly, with the interval between successive regenerations for any one spot considerably less than $1/10$ second. In the SILLIAC the interval varies from about $1/30$ second to about $1/50$ second according to the program running at the time.

There are two ways in which the nature of the program being performed can affect the regeneration of the memory. The first is that, because the memory spots can only be regenerated when the store is not in immediate use for reading out or receiving information, the amount of time actually available for this purpose depends upon the types of instructions being performed. For example, only one regeneration can be performed while an instruction of type 4 is being performed, while some 40 can be done during a **division** operation.

To understand the second way in which the type of program can effect the regeneration process we must first note that if a spot

is not regenerated in time it may be lost, not only through a gradual leakage away of the charge on the screen (a process which is purely a function of time between regenerations) but also because the reading or recording of information on adjacent spots on the screen has disturbed the one we are interested in. This disturbance is due to the clouds of secondary electrons which are emitted from the surface of the screen whenever a spot is recorded, read, or simply regenerated.

The circuitry of SILLIAC has been so designed that regeneration always takes place sufficiently often to prevent information being lost by the normal slow leakage of charge. It is also designed so that the spots adjacent to any one spot can be read or written a large number of times between two successive regenerations of the spot without destroying the information; but there is an upper limit to this number, known as the "Read Around Ratio" or RAR, and it is possible, in very short loops, for it to be exceeded.

The highest RAR which should be allowed in any program is 150. In order to ensure that the SILLIAC is reliable at this figure, it is tested every morning, and if necessary adjusted, so that it will run without error for a few minutes at a considerably higher figure.

CALCULATION OF THE RAR.

Although it is not possible to give a simple rule for determining which parts of a program are liable to yield a high RAR, without calculating the RAR for every part of the program; yet it is generally true that only the inner loops of a program, involving such simple operations as forming the sum of many store

locations, clearing a block of store locations, searching through a block of store locations, counting the number of binary one's in a group of words, etc., ever lead to a high RAR. The programmer should therefore, by use of the following rules, calculate the RAR for those parts of his program which are of this nature. After obtaining a little experience in doing this he will then find it easy to recognise any other sections of the program which may yield similar RAR's.

The calculation of the RAR for a program loop involves four separate steps:-

- 1) Calculation of the number of regenerations that take place during each operation of the loop. This figure is obtained by a simple formula described shortly.
- 2) From the number of regenerations R during each execution of the loop we obtain the number of times the loop is performed during the time in which the entire store is regenerated once (i.e. a complete "regeneration period"). This figure M, is given by the formula

$$M = 1024/R$$

If the number M obtained from this formula is greater than the maximum number of times that the loop will be repeated in immediate succession, then replace the value of M by this latter figure.

- 3) We now look at each of the storage locations which is operated upon during the execution of the loop. Each storage location which contains an order pair of the loop

is operated upon when that order pair is extracted and placed in the order register prior to execution. Each storage register containing a constant or variable which is used within the loop is operated upon whenever this quantity is read out from storage or stored. Some storage locations may be operated upon more than once during execution of the loop; if there are any such we should concentrate attention upon these and make a note of how often each one is used. For each store location affected by the loop we multiply the number of times it is used during each execution of the loop by the number M to obtain the number of times it is used during one complete regeneration period.

4) If any single storage location is operated upon more than 150 times during one full regeneration period, then we have exceeded the maximum allowed RAR for each of the locations adjacent to this one. If no one location has such a large figure we must next look to see whether two or more with smaller figures whose overall sum is greater than 150 are so situated that they are all adjacent to any one other storage location. As each one will affect this adjacent location when it is used, the RAR for this location will be exceeded. Although each storage location has six adjacent locations (see below for description of storage layout) it is very unlikely that any one loop will use each of these six equally often; hence we can take it as a rule of thumb that if no one of the storage locations used by the loop is operated upon more than 40 times during one regeneration period the maximum allowed RAR will not be exceeded.

Counting Regenerations. To count the number of regenerations that take place during each execution of a loop proceed as follows:-
Count 1 regeneration for every order pair in the loop. (The pair need not be complete; e.g. if the first order of the loop is the right hand member of a pair count the whole pair). Count 1 regeneration for every order in the loop of type 3-, 3L, 4, 5, N or J. Count 2 regenerations for every order of type F or L, and for every control transfer order actually obeyed. (A conditional control transfer for which the condition is not satisfied does not count). Count 3 regenerations for every order of type + or -. Count $n+1$ regenerations for every shift order involving a shift of n places.

The total of these figures is the value of R for the loop. Note that we have not given a count figure for input and output orders, multiplication and division orders, and magnetic tape orders. This is because the relatively long duration of these orders guarantees that any loop containing any one of them will not lead to RAR difficulty.

Storage Layout. As can be seen from an inspection of the monitor tube on the SILLIAC the information in the store is stored on spots on cathode ray tube screens in the form of a raster containing 32 rows and 32 columns. These 1024 storage locations are numbered from 0 in the top left corner, row by row, to 1023 in the bottom right hand corner. Hence it is obvious that storage locations whose addresses differ by 1 or by 32 are adjacent. However, in order to achieve a maximum density of spots on the screen, alternate rows

are staggered in a way as shown by the following diagram which depicts the top left hand corner of the screen. In this diagram each number represents the position of the storage location with that address.

0	1	2	3	4	5	6	
32	33	34	35	36	37		
64	65	66	67	68	69	70	
96	97	98	99	100	101		

For example, it can be seen from this diagram that location 0 has only two neighbours, 1 and 32; location 1 has four, 0, 2, 32 and 33; while 66 has six, 33, 34, 65, 67, 97 and 98.

This diagram suggests immediately a simple method for eliminating most of the difficulty that can arise from high RAR's. The method consists of using only the alternate locations 0,2,4,... etc for storage of intermediate results used in the inner loops of the program, and completely avoiding use of the locations from 32 to 63 for either instructions or data. If this plan is carried out there is no need to calculate the RAR for locations adjacent to the "working" locations 0, 2, ...etc., as these locations are not used.

It is not usually necessary to consider the arrangement of the storage locations when considering the RAR in locations adjacent to instructions. This is so because the instructions of any one loop usually are located in one sequence of successive store locations, hence, as a loop that produces a large RAR is

normally a short one, the instructions will be located in one line on the screen, or, partly in the right hand end of one line and partly in the left hand end of the next. Such instructions are therefore adjacent in the store layout only if they are adjacent in the program.

How to Remedy a Large RAR

As pointed out in the preceding section, the simplest way to remedy a too large RAR is simply to arrange that the store locations which are subject to the large RAR are not used. However this remedy is simple only when the location whose repeated use gives rise to the large RAR is used to contain a constant or a variable; the remedy is not so simple to apply when it is the locations adjacent to the program itself which suffer from the large RAR. Consequently, if a loop is found to have a RAR greater than 150 in locations adjacent to the instructions of the loop, it is generally best to modify the loop itself so that the RAR is reduced.

The simplest way to do this is to insert somewhere in the loop an order with a sufficiently long execution time. The best order to use for this purpose is a shift order as it is a simple matter to adjust the number of shifts so as to achieve the required RAR. Of course a place must be found in the loop where the contents of the arithmetic unit may be disturbed in this way. If no such place is available, then a pair of shift orders, such as

06 nF
16 nF

which leaves the arithmetic unit in the same condition (except for the overflow indicator which is now cleared) can be inserted anywhere.

If it is important that the running time of the program be not increased, then a different way must be used to reduce the RAR. A very powerful technique consists of writing the loop out to twice its original length, so that it is obeyed only half as frequently. To explain this latter idea let us consider an example of a loop used to clear a succession of storage locations. This loop may have originally been written in the form:-

p	<u>41</u>	<u>(q)</u> F
p+1	F5 42	pF pF
p+2	IO 32	rF pL

where an appropriate constant is store in rF. This loop takes 315 micro-seconds per repetition, and allows of 11 regenerations. Hence the number of repetitions per regeneration period is $1024/11 = 93$. Location p has 3 references per repetition, while p+1 and p+2 have 1 each. Hence both of the two storage locations which are adjacent to both p and p+1 will suffer a RAR of $4 \times 93 = 372$, a figure which is far too large.

Let us now re-write this loop in the form

p	41	(q)F
	41	(q+1)F
p+1	41	(q+2)F
	41	(q+3)F
p+2	L5	p+1L
	L4	tF
p+3	40	pL
	L4	tF
p+4	40	p+1L
	IO	rF
p+5	<u>36</u>	<u>pL</u>

where location tF contains the constant 002F 002F. The loop now allows of 22 regenerations per repetition, hence the number of repetitions per regeneration period is approximately 46.5. Location p has 2 references and p+1 has 3 references per repetition. Hence the two locations adjacent to both these two suffer a RAR of $5 \times 46.5 = 233$. By this change we have reduced the RAR considerably, but not enough, but it is obviously possible by making the loop even longer to reduce the RAR to the required figure of 150.

Note that the modified loop takes 685 micro-seconds per repetition, hence averages $685/4 = 171$ micro-seconds per storage location cleared. Thus our change has also resulted in a faster program! If we were satisfied with the speed of the original loop we can reduce the speed of the second one to the same figure by including an order which takes 575 micro-seconds: a multiplication order would be about right, as also would be the order 00 36F (16 micro-seconds per shift). Whichever of these two we use, we are allowing an additional $575/18 = 32$ regenerations per repetition, thereby increasing the total now to 54. This reduces the RAR from 233 to 95. Thus we have solved the problem of reducing the RAR to a figure less than 150 without affecting the operating speed of the program.

To provide another example showing how the RAR can be reduced without sacrificing speed consider the case of a loop used to form the sum of a sequence of storage locations. Such a loop will be a little longer than the one we have considered above, but it can still exceed the legal maximum RAR of 150. For example

p	<u>L5</u>	<u>(q)</u> F		p	3L	iF
			or		L5	qF
p+1	L4	sF		p+1	L4	sF
	40	sF			40	sF
p+2	F5	pL		p+2	F5	iF
	42	pL			40	iF
p+3	L0	rF		p+3	L0	rF
	32	pL			36	pL

We have given here two different ways of doing the same operation, the first directly modifies the address of an order, while the second uses an index order. Let us compare these two loops.

The first loop allows 16 regenerations per repetition, hence is repeated 64 times in any regeneration period. The second allows 17 regenerations, hence is repeated (approximately) 60 times in a regeneration period. In the first, locations pL and sF have the largest number of references; pL has $3 \times 64 = 192$ references per regeneration period, sF has $2 \times 64 = 128$, while p+1L has 64. Hence either of the two locations adjacent to both pL and p+1L suffers a RAR of 256. This figure is too high and must be reduced; we must also make sure that location sF is not so placed that it and the loop have common neighbours. To reduce the RAR of this loop without increasing the operation time we must expand it so that two successive storage locations are added in each repetition of the loop. For example we code it as:

p	L5	(q)F	
	L4	(q+1)F	
p+1	L4	sF	
	40	sF	
p+2	L5	pL	
	L4	tF	
p+3	40	pL	
	L0	rF	
p+4	<u>36</u>	<u>pL</u>	IV-10

This change reduces the RAR by only a small amount, and it is still too large; but because we now have a faster operation (615 micro-seconds for each pair of locations added, as compared with 465 for each one) we can slow the loop down (by inserting a shift order) sufficiently to reduce the RAR to less than 150.

In the second loop we have the large RAR figures of $3 \times 60 = 180$ in locations adjacent to iF , and $2 \times 60 = 120$ in locations adjacent to sF , also $2 \times 60 = 120$ in all locations adjacent to any two of the instruction pairs. So we must reduce the number of references to location iF during each regeneration period. As in the case of the first loop, we can arrange that two store locations are added each time; e.g. by starting the loop with

p	3L	iF
	L5	qF
p+1	3L	iF
	L4	q+1F

and by adding 2 to i each time. This increases the number of references to iF from 3 to 4, hence the RAR is now $4 \times (1024/24) = 171$ whereas before it was $3 \times (1024/17) = 181$. But now we can reduce the RAR to below 150 by inserting a shift order which gives just 3 shifts, and still have a loop which is faster than the original one. Of course it is important that we arrange the layout of the store so that locations iF and sF can have no common neighbours with each other nor with the program loop.

INDEX

-A-

A register, 2-4
Accumulator, 2-4
Accuracy, 6-6
Add from Q, 2-7, 3-44
Addend, 2-5
Addition, 2-5, 2-6, 3-48
Address, 1-3, 3-2
 as constant, 7-7
 digits, 3-15
 fixed, 5-2
 relative, 5-2
And Operation, 7-18
AQ, 3-18
Arithmetic unit, 1-2, 2-4
Augend, 2-5

-B-

Binary
 chopping, 7-13
 -coded decimal, 9-3
 digit, 2-1
 point, 2-1
 point shifted, 6-1
 switch, 7-4
 system, 2-1, App. I
Blocking order, 8-7
Blunders, 8-5
 list of typical, 8-6
Bootstrap, 5-12
 start, 5-13

-C-

Checking, 8-1
Check point, 8-7
Closed subroutine, 4-2
 entry, 4-3
Coding tricks, 7-1
Comparing of tapes, 9-6
Complement, 2-2
Conditional transfer, 3-7, 3-25
Constants, orders and addresses as, 7-7
Control counter, 8-5
Control transfer, 3-7
 reversed, 7-3

Control unit, 1-4
Convergence criteria, 7-11
Cycle, 1-5
 resetting and starting, 7-8

-D-

Decimal addresses, 5-5
Decimal order input, 5-1
Definition of terms, 14-1
Digital, 1-2
Digitwise complement, 2-2
Directive, 5-4
Division, 2-12, 3-29
 "double precision", 2-17
 hangup, 2-15
 of integers, 2-19
 remainder, 2-14, 7-13

-E-

Entry to D.O.I., 5-36
Equals operation, 7-18
Execution of orders, 3-6
Extract, 3-46

-F-

Fifth hole, 9-4
Fixed addresses, 5-2
Fixed point, 2-1
Floating addresses, 5-25, 5-36
Floating point routines, 6-9
Format characters, 9-3
Fractions, input of, 5-6
Function digits, 3-17

-G-

Glossary, 14-1

-I-

Increment add, 2-7, 3-47
 from Q, 2-7, 3-43
Indexing orders, 3-10, 3-25
Information sources, 8-5

Input, 9-1
Input-output orders, 3-4, 3-31, 3-36
Input of decimal fraction with D.O.I., 5-6
Integer operations, 2-18
Interchange of numbers, 7-10
Interlude, 5-11
Interpretive routine, 4-12

-L-

Leapfrog, 2-16
Left shift, 2-3, 2-8, 3-19
Letter printing, 9-4
Library, 13-1
Link, 4-4
Location, 1-3
Logical operations, 3-57, 7-17
Logical product, 3-46
Loop, 8-7

-M-

Marking, 7-12
Memory, 1-2
 to Q, 2-16, 3-28
Mistakes, 8-1
Modification of routines, 5-16
Multiplication, 2-10, 3-30

-N-

Negative numbers, 2-2
Not equivalent operation, 7-18
Null orders, 3-12, 3-25
Number register, 2-5

-O-

One-address code, 3-2
Or operation, 7-18
Order, 1-4
 assembly, 5-5
 blocking, 8-7
 code, 3-1
 as a constant, 7-7
 counter, 3-9, 8-5
 execution time, 10-2
 pair, 3-3
 types, 3-13, 3-14
 variants, 3-13

Ordvac, 1-1
Output, 9-1
Output orders, 3-4, 3-31, 3-36
Overflow, 2-3, 3-8

-P-

Page printer, 9-1
Parameters,
 preset, 4-13, 5-7
 program, 4-9
Pierce operation, 7-18
Polynomial evaluation, 7-15
Post mortem routine, 8-9
Product summation, 7-1
Program, 1-4
 interruption, 3-9
Pseudo orders, 7-7

-Q-

Q register, 2-4
Quotient register, 2-4
 for interchanges, 7-10

-R-

R_3 , 3-4
 R^3 , 2-4
Relative address, 5-2
Remainder, 2-12
 in integer division, 7-13
Reperforation, 9-6
Residue, 2-13
Right shift, 2-3, 2-8, 3-21
Routine, 1-4
 address search, 8-12
 check point, 8-13
 closed, 4-2
 entry to, 4-3
 exit from, 4-4
 interpretive, 4-12
 post mortem, 8-9
 sequence checking, 8-9
Running time, 10-1
 simple formula, 10-4

-S-

Scaling, 6-1
 adjustable, 6-9
 continuous, 6-9
Scheffer stroke operation, 7-18
S directives, 5-20
Sexadecimal, 3-3
Shift, 2-3, 2-8
Sign digit, 2-1
Starting a program, 5-6
Stop orders, 3-12
Stopping tape, 5-11
Store, 2-16, 3-27, 3-45
Structure of orders, 3-2
Subtraction, 2-6, 3-48
Sum check facility, 5-22

-T-

Tagged addresses, 5-25 to 5-36
Tape code, 9-1,
 complete, 9-2
Tape editing, 9-4
Teleprinter code, 3-39
Teletype equipment, 9-7
Testing for 0 and -1, 7-6
 for $1/2$, 7-11
Transfer of control, 1-5, 3-7

-U-

Unconditional transfer, 3-7, 3-23

-W-

White switch start, 3-23